5-2013

Evolutionary Optimization Algorithms for Nonlinear Systems

Ashish Raj

EVOLUTIONARY OPTIMIZATION ALGORITHMS FOR NONLINEAR SYSTEMS

by

Ashish Raj

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Electrical Engineering

Approved:

Dr. Edmund A. Spencer Major Professor	Dr. Chris Winstead Committee Member
Dr. Thidapat Chantem Committee Member	Dr. Mark R. McLellan Vice President for Research and Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2013

Abstract

Evolutionary Optimization Algorithms for Nonlinear Systems

by

Ashish Raj, Master of Science

Utah State University, 2013

Major Professor: Dr. Edmund A. Spencer
Department: Electrical and Computer Engineering

Many real world problems in science and engineering can be treated as optimization problems with multiple objectives or criteria. The demand for fast and robust stochastic algorithms to cater to the optimization needs is very high. When the cost function for the problem is nonlinear and non-differentiable, direct search approaches are the methods of choice. Many such approaches use the greedy criterion, which is based on accepting the new parameter vector only if it reduces the value of the cost function. This could result in fast convergence, but also in misconvergence where it could lead the vectors to get trapped in local minima. Inherently, parallel search techniques have more exploratory power. These techniques discourage premature convergence and consequently, there are some candidate solution vectors which do not converge to the global minimum solution at any point of time. Rather, they constantly explore the whole search space for other possible solutions.

In this thesis, we concentrate on benchmarking three popular algorithms: Real-valued Genetic Algorithm (RGA), Particle Swarm Optimization (PSO), and Differential Evolution (DE). The DE algorithm is found to out-perform the other algorithms in fast convergence and in attaining low-cost function values. The DE algorithm is selected and used to build a model for forecasting auroral oval boundaries during a solar storm event. This is compared against an established model by Feldstein and Starkov. As an extended study, the ability of

the DE is further put into test in another example of a nonlinear system study, by using it to study and design phase-locked loop circuits. In particular, the algorithm is used to obtain circuit parameters when frequency steps are applied at the input at particular instances.

(90 pages)

Public Abstract

Evolutionary Optimization Algorithms for Nonlinear Systems

by

Ashish Raj, Master of Science

Utah State University, 2013

Major Professor: Dr. Edmund A. Spencer
Department: Electrical and Computer Engineering

In the real world, we encounter a number of problems which require iterative methods rather than heuristic approaches to solve them. Not every problem can be solved with a definitive method. Optimization algorithms come to the aid of such instances. These algorithms carry out multiple iterations or generations to try to achieve the lowest value of a cost function. The demand for fast and robust stochastic algorithms to cater to the optimization needs is very high. The faster the convergence to a low value of the cost function, the better the algorithm is. This is attained in greedy criterion approaches, where the new parameter vector is accepted only if it reduces the value of the cost function. But this may also lead in misconvergence where it could lead the vectors to get trapped in local minima. Parallel search techniques have more exploratory power. So, depending on the application, different suitable techniques are used.

This thesis mainly concentrates on benchmarking three popular algorithms: Real-valued Genetic Algorithm (RGA), Particle Swarm Optimization (PSO), and Differential Evolution (DE). The DE algorithm is found to out-perform the other algorithms in fast convergence and in attaining low-cost function values. The DE algorithm is selected and used to build a model for forecasting auroral oval boundaries during a solar storm event. This is compared against an established model by Feldstein and Starkov. As an extended

study, the ability of the DE is further put into test in another example of a nonlinear system study, by using it to study and design phase-locked loop circuits. In particular, the algorithm is used to obtain circuit parameters when frequency steps are applied at the input at particular instances.

To my beloved parents and dearest brother...

Acknowledgments

This research project has been a very elucidative and scholastic experience for me. Foremost, I would like to thank my advisor and mentor, Dr. Edmund Spencer, for believing in me and taking me as his student, for introducing me to the topic and setting a standard of diligence with my work, for his unwaivering support and willingness to listen, the list is endless. I always wondered how it would be like to meet a teacher and a friend in the same person; I now finally have. Thank you, Dr. Spencer, for being one of my biggest sources of inspiration.

I would also like to thank Dr. Chris Winstead and Dr. Tam Chantem for their insightful suggestions and technical guidance along the trajectory of my work. Their earnest support is truly commendable, and without the invaluable motivation and encouragement that they provided, it would never have led to a successful completion of my research work.

Special thanks to my fellow colleagues of the Center for Space Engineering (CSE) lab: Swadesh Patra, Tushar Andriyas, and Prajwal Kasturi. It was an absolute pleasure working alongside you guys; I attribute much of my success to you all. You have been instrumental in making my academic life a huge success story. I was showered with tremendous technical and emotional support whenever I was in need; thank you.

I have received immense financial support from the English Department. I would like to deeply thank the Suite Lab managers, staff, and colleagues, for providing me with a wonderful opportunity to gain a whole new experience here at USU.

Just like on countless other occasions, both on paper or verbally, I once again feel proud to laud my parents and my little brother here, for their eternal love and for being my greatest strength, ever since I entered this world.

This acknowledgment section can never be complete without extending my heart-felt gratitude to all of my friends in Logan. I have been lucky enough to meet some of the most wonderful and delightful people: Rajee, Debrup, Ravi, Renil, Swathi, Sulochan, Pratima, to name a few. You have helped me in maintaining a delicate balance between stress and

fun throughout. You have shown and taught me so many things that I would never have come across in any book. I shall be forever indebted to you. I have truly been blessed with loads of invaluable memories that shall be cherished forever.

Ashish Raj

Contents

List of Tables

List of Figures

Chapter 1

Introduction

We encounter complex computational problems every day, and they need to be analyzed effectively and efficiently. For this purpose, optimization methods are employed. Many mathematical models have been built for this purpose of iterative optimization. These techniques were inspired by nature itself, as such behaviors are observed in biological and natural processes like evolution and in genetic study. Over the many millions of years, all the different species had to adapt their physical structures to adapt to the environment they have been living in. The different species have to explore and search for food, cover large spaces where the probability of finding these resources is high, and finally converge at these locations. Similar analogies, including the usage of terms in a metamorphic way, have been applied to these models for searching and optimization.

These models can be broadly classified under two main categories: those corresponding to individual behavior and those using collective behavior. For the former, simple gradient strategies are used since they are the most obvious. But in cases of functions that have multiple summits, when the strategy employed is used to find the summit, the model could end up pointing to a secondary summit that is actually lower than the first one. But this limitation is avoided to a great extent in the case of the latter. Here, both the size of the group as well as the structure contributes to the collective optimization. Such a technique is accurately demonstrated in techniques such as the particle swarm optimization (PSO) [1], ant colony optimization (ACO), bees' algorithm, bacterial foraging optimization (BFO), intelligent water drops (IWD) algorithm, etc.

Evolutionary computation uses iterative progress, such as growth or development in a population. Parallel processing is carried out on this and a guided random search is implemented to achieve the desired result. This method shows the strong influence of Darwinian

principles. Ever since this idea came into light, the field of nature-inspired meta-heuristics is dominated by the evolutionary algorithms such as genetic algorithms (GA) [2], evolutionary programming (EP), evolutionary strategies (ES), genetic programming (GP), harmony search (HS), memetic algorithm (MA), differential search (DS), bacteriologic algorithms (BA), gaussian adaptation (NA), cultural algorithm (CA), differential evolution (DE) [3], etc.

In this work, we first venture into the world of stochastic optimization algorithms in Chapter 2. We mainly focus on studying the genetic algorithm (GA), real-valued genetic algorithm (RGA), particle swarm optimization (PSO) and differential evolution (DE), and the steps that are involved in each algorithm that drive any particular problem at study to arrive at an optimum solution. These algorithms are chosen because they are popular algorithms which are used in a wide variety of applications, and they are good individual representatives of collective optimization and evolutionary computation techniques. In Chapter 3, we proceed to benchmark these algorithms against six standard test functions. This gives a good idea on which algorithm could be chosen for a particular application depending on the quality of performance. We do not consider the GA here, as the RGA is essentially a modified version of the GA, which uses real values instead of binary values. We document the performance results of the RGA, PSO, DE and a variant of the DE, and compare the performance of each against the other. In Chapter 4, we discuss the methodology of obtaining a model to forecast the auroral oval boundaries (equator-ward and pole-ward). This model will be trained on available data on the British Antarctic Survey website and the Data Analysis Center for Geomagnetism and Space Magnetism of Kyoto University. Finally, in Chapter 5, we further try to explore the versatility of the DE algorithm, and apply it in phase-locked loop circuits to obtain optimum input parameters or coefficients.

Chapter 2

Optimization Algorithms

Optimization algorithms are tools which aid in finding the best solution from the pool of available alternate candidates. In essence, they try to maximize or minimize a cost function by choosing inputs parameters from an allowed search space and evaluating the cost function in systematic steps. The following sections discuss a few of these typical algorithms.

2.1 Genetic Algorithm (GA)

Charles Darwin published and established various theories in the field of evolution with compelling evidence. He concluded that the struggle of existence in nature led to natural selection. This resulted in the diversity of life, all of which he elaborately documented in his book, On the Origin of Species in 1859. But little did he know back then, that one day, his principles would be actually applied in optimization algorithms and programming, to converge to the most optimum solutions in linear and nonlinear problems or cost functions. In fact, the natural selection and genetic theories have inspired these algorithms to such an extent that even the terminologies used are the same.

GAs encode the decision variables of a search problem into finite-length strings of alphabets of certain cardinality. The strings which are candidate solutions to the search problem are referred to as chromosomes, the alphabets are referred to as genes, and the values of genes are called alleles [4]. The population size defines the candidate pool from which the solutions are chosen. Choosing the size of this pool is critical for a GA to function efficiently. Selecting a large candidate pool could increase the computational times to a great degree, whereas choosing a small population could lead to premature convergence and sub-standard solutions. So, there is a trade-off between computational time and good

solutions. A normal genetic algorithm can be divided into the following sequence of steps, which is represented in Fig. 2.1:

1. Initialization,
2. Evaluation,
3. Selection,
4. Recombination,
5. Mutation,
6. Replacement,
7. Termination.

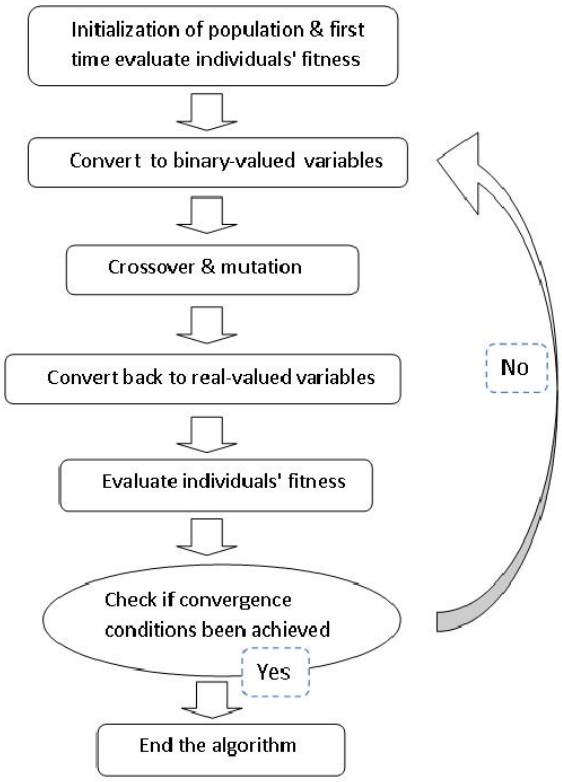

Fig. 2.1: Flow chart of the genetic algorithm.

2.1.1 Initialization

The initial population of candidate solutions is usually generated randomly across the search space. This step is crucial to the algorithm's functioning, because this is the step where the limits are defined for every parameter of every individual member in the population. Hence, the search space has to cover all the possible values that the parameter can hold. The population is usually of the form 2^n, where n is a positive integer. Every individual parameter in every individual member, is denoted as var^{ij}, where i $\epsilon[1, ..., N_{var}]$ denotes the index of the N_{var} parameters or dimensions that are present in the function, and j $\epsilon[1, ..., pop]$ denotes the index of the *pop* members or variables in the population pool. Every var^{ij} is initializes as

$$var^{ij} = var^{ij}_{min} + \left(\frac{var^{ij}_{max} - var^{ij}_{min}}{2^n} \right) m^{ij}, \tag{2.1}$$

where

var^{ij}_{min} is the minimum limit of the parameter,

var^{ij}_{max} is the maximum limit of the parameter,

m^{ij} is an integer of the range [0, 2^n-1].

2.1.2 Evaluation

The main aim of the evaluation step is to define a cost function to meet the application demands of the problem in focus. This fitness function could be a lowest mean square error (MSE) or a normalized L2 norm of the simulated value against the target value. It actually depends on the application itself. Depending on the type of problem under study, the design of the fitness function is important for the GA structure to carry out efficient optimization, since fitness is the measure of the goodness of an individual. The GA searches the input domain of the subject program for suitable test cases to kill a given mutant. The guidance for this is provided by the fitness function which assigns a non-negative cost to each candidate input value. An input with a zero cost is a test case that kills the mutant. Some of the properties of a good cost function are that it should be non-negative, continuous, no

fixed costs, etc.

2.1.3 Selection

Selection introduces the influence of the fitness function to the GA process. Selection must utilize the fitness of a given individual. However, selection cannot be based solely on choosing the best individual, because the best individual may not be very close to the optimal solution. Instead, some chance that relatively unfit individuals are selected must be preserved, to ensure that genes carried by these unfit individuals are not lost prematurely from the population. In general, selection involves a mechanism relating an individuals fitness to the average fitness of the population.

A number of selection strategies have been developed and utilized for the GA optimization. They are generally classified as either stochastic or deterministic. The selection strategies include processes like population decimation, proportionate selection or roulette-wheel selection, tournament selection, etc. The former is a deterministic strategy, while the latter two are stochastic. Usually, stochastic methods are better suited for the algorithm because it is always preferred that some of the less fit solutions are retained in the population as the iterations proceed. This keeps the diversity of the population large and avoids premature convergence. In this work, half of the existing population is kept and the other half is discarded in every generation. The selection implement is also based on a fitness-based process by ranking the fitness of the individuals.

2.1.4 Recombination

Recombination or crossover combines parts of two or more parental solutions to create new, possibly better solutions (i.e., offspring). There are many ways of accomplishing this, and competent performance depends on a properly designed recombination mechanism. The offspring under recombination will not be identical to any particular parent and will instead combine parental traits in a novel manner. There are many variations of crossover.

The simplest of these is single-point crossover. In single-point crossover, shown in Fig. 2.2, if $p > p_{cross}$, a random location in the parent's chromosomes is selected. The

portion of the chromosome preceding the selected point is copied from parent number 1 to child number 1, and from parent number 2 to child number 2. The portion of the chromosome of parent number 1 following the randomly selected point is placed in the corresponding positions in child number 2, and vice versa for the remaining portion of parent number 2's chromosome. If $p < p_{cross}$, the entire chromosome of parent number 1 is copied into child number 1, and similarly for parent number 2 and child number 2. p and p_{cross} denote the probabilities.

2.1.5 Mutation

While recombination operates on two or more parental chromosomes, mutation locally but randomly modifies a solution. Again, there are many variations of mutation, but it usually involves one or more changes being made to an individual's trait or traits. In other words, mutation performs a random walk in the vicinity of a candidate solution.

The mutation operator provides a means for exploring portions of the solution surface that are not represented in the genetic makeup of the current population. In mutation, if $p > p_{mutation}$, an element in the string making up the chromosome is randomly selected and changed. In the case of binary coding, this amounts to selecting a bit from the chromosome string and inverting it. In other words, a "1" becomes a "0" and a "0" becomes a "1." If higher- order alphabets are used, slightly more complicated forms of mutation are required. p and $p_{mutation}$ denote the probabilities.

Generally, it has been shown that mutation should occur with a low probability, usually on the order of $p_{mutation}$ = 0.01 - 0.1. The action of the mutation operator is illustrated in Fig. 2.3, which shows a randomly selected element of binary 1 value in a chromosome being changed to binary 0 value.

2.1.6 Replacement

The offspring population created by selection, recombination, and mutation replaces the original parental population. Many replacement techniques such as elitist replacement, generation-wise replacement, and steady-state replacement methods are used in GAs.

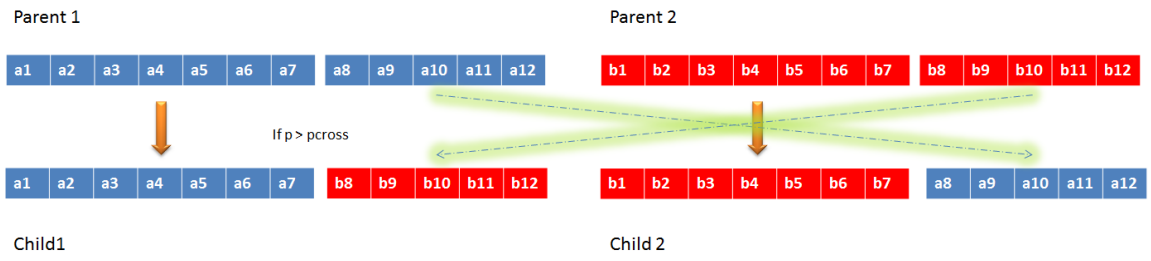

Fig. 2.2: Single-point crossover.

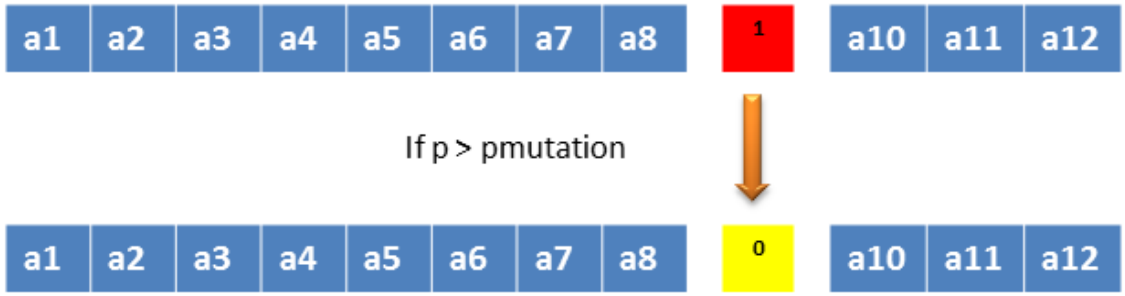

Fig. 2.3: Mutation.

2.1.7 Termination

The reproduction process will keep repeating until one of the conditions to end the algorithm has been achieved. Usually, the ending criteria will be one of the following:

1. A solution is found that satisfies the minimum criteria;
2. A fixed number of generations is reached;
3. Allocated budget (computation time/money) is reached;
4. The highest ranking solution's fitness is reaching or has reached a plateau such that successive iterations no longer produce better results;
5. Manual inspection;
6. Combinations of the above.

2.2 Real-Valued Genetic Algorithm (RGA)

In its originally conceived form, the genetic algorithm involves binary genetic sequences that are converted from real-valued variables. This results in extra effort of the conversion between binary and real values. The resolution of accuracy of every individual solution will be difficult to decide. The RGA can handle real-valued variables directly while processing both the crossover and mutation process. By using an RGA, every individual solution retains a machine specific bit precision.

The flow chart of an RGA is demonstrated in Fig. 2.4. As we can see, here too, the initial solutions are a data-set of random values chosen from within the range of the variables. Next, the crossover produces off-springs that retain the good properties of their parents, that are different from their parents as well as each other and are produced randomly. The mutation process of RGA has to produce a random new individual in its particular variable range. Finally, the individuals' fitness values are evaluated and the process of crossover and mutation is repeated if convergence has not been achieved yet. The main skeleton of the RGA follows the GA. They are discussed in the following sub-sections.

2.2.1 Initialization

There is no issue of resolution consideration with the RGA, because it deals with only real values. So the resolution is set by machine precision. The values are initialized within the search space in a random fashion, as follows:

$$var^{ij} = var^{ij}_{min} + \left(var^{ij}_{max} - var^{ij}_{min}\right) m^{ij}, \tag{2.2}$$

where

var^{ij}_{min} is the minimum limit of the parameter,

var^{ij}_{max} is the maximum limit of the parameter,

m^{ij} is an integer of the range [0, 1].

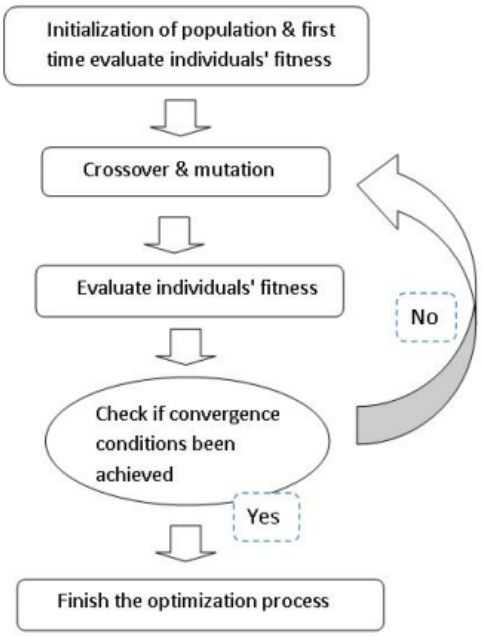

Fig. 2.4: Flow chart of the real-valued genetic algorithm.

2.2.2 Crossover

In the crossover procedure of the RGA, two solutions called parents are randomly selected from a mating pool. The parents are to do the crossover process if the crossover probability is larger than a random value between 0 and 1. Then the genetic operation of sexual recombination is applied to the pairs of parameters of the parents. In general, there are different genetic operation modes of sexual recombination. There are various types of crossover methods [5].

The RGA crossover follows three main rules. First, the children produced as a result of crossover should retain the good characteristics of the parents. Second, the children have to be different from their parents and from each other as well. Third, the children have to be produced randomly, which follows a wide-ranged distribution, like the Gaussian distribution for example. The real-valued crossover can be visualized as follows:

$$S_{1j} = (1 + r_1)(k_1 s_{1j} + (1 - k_1)s_{2j}), \tag{2.3}$$

$$S_{2j} = (1 + r_2)(k_2 s_{1j} + (1 - k_2)s_{2j}), \tag{2.4}$$

where

k_1, k_2, r_1 and r_2 are random numbers in the range [0, 1];

s_{1j} and s_{2j} are the j'th element of the parents s_1 and s_2, respectively;

S_{1j} and S_{2j} are the j'th element of the children of s_1 and s_2, respectively.

The range of the children data will cover a larger space by using $(1 + r_1)$ and $(1 + r_2)$, instead of only covering the range between parents data. However, this might bring up an offspring value which is born outside of the range. If this happens, the child is reproduced within the range, by using equations (2.3) and (2.4) and eliminating the $(1 + r_1)$ and $(1 + r_2)$ terms.

2.2.3 Mutation

The mutation process of RGA has to produce a random new individual in its particular variable range, the same way as in the first generation individual in equation (2.2). Because of the mechanism of the mutation process, the mutation rate values of the RGA has different scales from the binary GA.

2.3 Particle Swarm Optimization (PSO)

2.3.1 Background

Particle swarm optimization (PSO) [6] is a population based stochastic optimization technique developed by Dr. Eberhart and Dr. Kennedy in 1995, inspired by the social behavior of bird flocking or fish schooling. It is a computational method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. The swarm mentality can be closely related to the bird flocking behavior. For instance, let us suppose a flock of birds take off in search of food in different directions. Let us also suppose there is only one piece of food present in this area under consideration. All the birds do not know where the food is. But they know how far the food is in each

iteration. The effective strategy is to follow the bird which is nearest to the food.

PSO learned from the scenario and used it to solve the optimization problems. In PSO, each single solution is a "bird" in the search space, which is called a "particle." All of the particles have fitness values which are evaluated by the fitness function to be optimized, and have velocities which direct the flying of the particles. The particles fly through the problem space by following the current optimum particles.

2.3.2 The Algorithm

The algorithm functions by first creating a population of candidate solutions. Each entity in the population is treated as a particle, with a certain position and velocity. These particles are moved around in the search-space to obtain the best fitness value for a corresponding function that needs to be optimized. The movement of the particles is based on two factors; the locally known best known position of the particles, as well as the global best position that has been updated after every iteration by all the particles together.

Every individual particle is initialized in a search space. The i'th particle is represented as $X_i = (x_{i1}, x_{i2}, ..., x_{iD})$. The best previous position (the position giving the best fitness value) of the i'th particle is recorded and represented as $P_i = (p_{i1}, p_{i2}, ..., p_{iD})$. The index of the best particle among all the particles in the population is represented by the symbol g. The rate of the position change (velocity) for particle i is represented as $V_i = (v_{i1}, v_{i2}, ..., v_{iD})$. The particles are manipulated according to the following equations:

$$v_{id} = c_0 * v_{id} + c1 * rand1() * (p_{id} - x_{id}) + c2 * rand2() * (p_{gd} - x_{id}), \tag{2.5}$$

$$x_{id} = x_{id} + v_{id}. \tag{2.6}$$

The inertia weight c_0 is employed to control the impact of the previous history of velocities on the current velocity, thereby influencing the trade-off between global (wide-ranging) and local (fine-grained) exploration abilities of the "flying points." A larger inertia weight facilitates global exploration (searching new areas) while a smaller inertia weight tends to facilitate local exploration to free-tune the current search area. Suitable selection

of the inertia weight provides a balance between global and local exploration abilities and thus requires fewer iterations on average to find the optimum [7]. c_1 and c_2 are two positive constants, and rand1() and rand2() are two random functions generating values in the range [0 1].

Unlike the genetic algorithms, the PSO does not have any selection process. All the members of the population pool are kept intact. It is only the velocities of each of these particles that are updated through their flying experiences. This is updated according to the individual particle's own previous best position and also the previous best position of its companions. Figure 2.5 represents the different displacements of a particle. The particles fly with the updated velocities. PSO is the only evolutionary algorithm that does not implement survival of the fittest technique [8]. The PSO algorithm can be summarized as follows:

1. Initialize every particle in the population pool within the search space of initial velocities;
2. Calculate the fitness value of each of these particles;
3. If this value is better than its best fitness value recorded until this time, which is also called as population best (pb), then set the current value as the new pb;
4. The global best (gb) is selected from the candidates, which is the best fitness value of all of them in the pool. This is updated in every iteration after this comparison as the new gb;
5. Calculate the particle velocity according to equation (2.5);
6. Update the particle position according to equation (2.6);
7. Carry out steps 2 - 5 until minimum error criteria is attained.

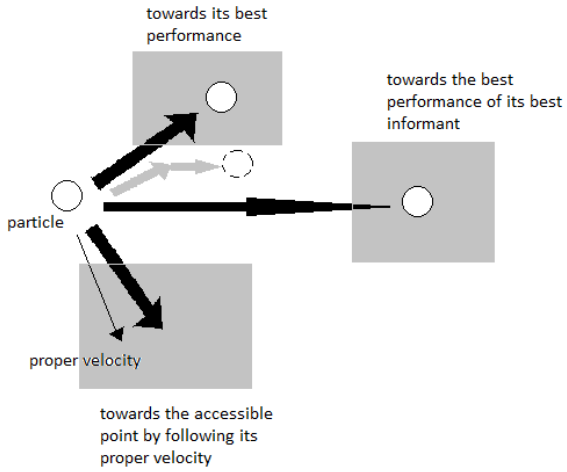

Fig. 2.5: Particle displacements.

2.4 Differential Evolution (DE)

2.4.1 Background

Differential evolution (DE) is arguably one of the most powerful stochastic real-parameter optimization algorithms in current use. The DE [9–13] algorithm emerged as a very competitive form of evolutionary computing more than a decade ago. The first written article on DE appeared as a technical report by R. Storn and K. V. Price in 1995 [11]. It is capable of handling non-differentiable, nonlinear, and multimodal objective functions. DE has been used to train neural networks having real and constrained integer weights. Its simplicity and straightforwardness in implementation, excellent performance, fewer parameters involved, and low space complexity, has made DE one of the most popular and powerful tool in the field of optimization. It works through a simple cycle of stages, presented in Fig. 2.6.

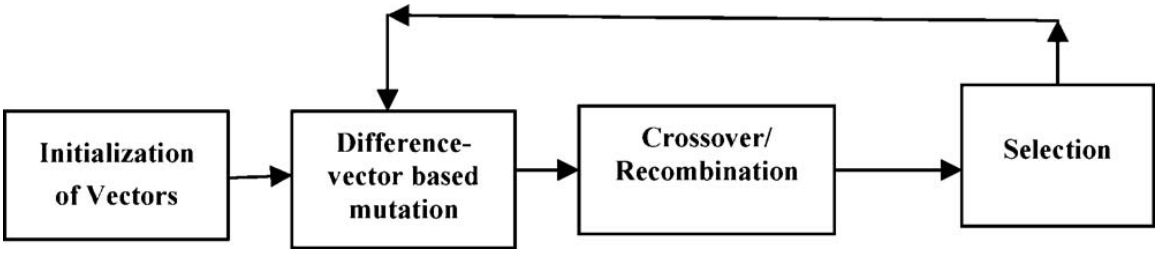

Fig. 2.6: Main stages of the DE algorithm.

2.4.2 The Algorithm

DE searches for a global optimum point in a D-dimensional real parameter space. The algorithm starts by creating an initial population of NP D-dimensional real-valued parameter vectors. The vectors are randomly initialized within a certain range that the values of the parameters are restricted to. The parent vectors of the current generation are known as the target vectors. They are mutated to produce donor vectors, by scaling the difference of two randomly chosen vectors from the population and adding the result to a third random vector picked from the same population. The trial vectors are then generated by binomial or exponential crossover methods. Finally, there is selection done between the target and the trial vector populations for the next generation, based on the values of the objective function that needs to be minimized. The algorithm can be documented according to the stages discussed in the following sections.

2.4.2.1 Initialization of the Parameter Vectors

The DE algorithm begins with a randomly initiated population of NP D-dimensional real-valued parameter vectors. Each vector, also known as genome/chromosome, forms a candidate solution to the multi-dimensional optimization problem. Subsequent generations in DE are denoted by G = 0, 1, ... , G_{max}. The i'th vector of the population at the current generation is denoted as

$$\vec{X}_{i,G} = [x_{1,i,G}, x_{2,i,G}, ..., x_{D,i,G}]. \tag{2.7}$$

Every parameter in the population is constrained to a certain search space, and they cannot exceed either ends of the limits. The minimum and maximum bounds are denoted as $\vec{X}_{min} = \{x_{1,min}, x_{2,min}, ..., x_{D,min}\}$ and $\vec{X}_{max} = \{x_{1,max}, x_{2,max}, ..., x_{D,max}\}$, respectively. Therefore, the j'th component of the i'th vector is initialized as

$$x_{j,i,0} = x_{j,min} + rand_{i,j}[0,1] \cdot (x_{j,max} - x_{j,min}), \tag{2.8}$$

where $0 \leq rand_{i,j}[0,1] \leq 1$.

2.4.2.2 Mutation with Difference Vectors

Mutation is the change or perturbation with a random element. In DE-literature, a parent vector from the current generation is called *target* vector, a mutant vector obtained through the differential mutation operation is known as *donor* vector and finally an offspring formed by recombining the donor with the target vector is called *trial* vector. To create the donor vector for each i'th target vector from the current population, three other distinct parameter vectors, $\vec{X}_{r_1^i}$, $\vec{X}_{r_2^i}$, and $\vec{X}_{r_3^i}$ are sampled randomly from the current population. The indices r_1^i, r_2^i, and r_3^i are mutually exclusive integers randomly chosen from the range [1, NP], which are also different from the base vector index i. These indices are randomly generated once for each mutant vector. The difference of any two of these three vectors is scaled by a scalar number F (that typically lies in the interval [0.4, 1]) and the scaled difference is added to the third one whence the donor vector $\vec{V}_{i,G}$ is obtained. The process can be expressed as

$$\vec{V}_{i,G} = \vec{X}_{r_1^i,G} + F \cdot (\vec{X}_{r_2^i,G} - \vec{X}_{r_3^i,G}). \tag{2.9}$$

This process is depicted in Fig. 2.7, which also shows the constant cost contours of an arbitrary objective function.

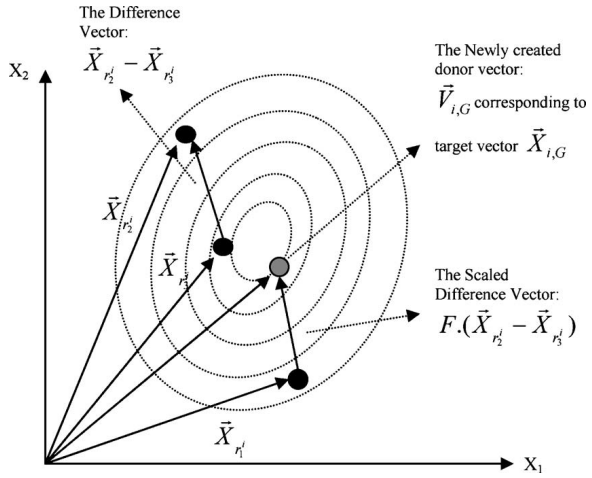

Fig. 2.7: Illustrating a simple DE mutation scheme in 2-D parametric space.

2.4.2.3 Crossover

After generating the donor vector through mutation, the crossover step is carried out to enhance the diversity of the population pool. The donor vector exchanges its components with the target vector $\vec{X}_{i,G}$ to form the trial vector $\vec{U}_{i,G} = [u_{1,i,G}, u_{2,i,G}, ..., u_{D,i,G}]$. The DE family of algorithms can use two kinds of crossover methods - exponential (or two-point modulo) and binomial (or uniform) [3]. In exponential crossover, an integer n is randomly chosen among the numbers [1, D]. This integer acts as a starting point in the target vector, from where the crossover or exchange of components with the donor vector starts. Another integer L is chosen from the interval [1, D]. L denotes the number of components the donor vector actually contributes to the target vector. After choosing n and L the trial vector is obtained as

$$\begin{aligned} u_{j,i,G} &= v_{j,i,G} \qquad \text{for j} = \langle n \rangle_D, \langle n+1 \rangle_D, ..., \langle n+L-1 \rangle_D \\ &= x_{j,i,G} \qquad \text{for all other j } \epsilon \text{ [1, D]}, \end{aligned} \tag{2.10}$$

where the angular brackets $\langle \rangle_D$ denote a modulo function with modulus D.

Cr is called the crossover rate. It is a control parameter, just like F. The binomial crossover scheme maybe outlined as

$$\begin{aligned} u_{j,i,G} &= v_{j,i,G} \qquad \text{if } (rand_{i,j}[0,1] \leq Cr or j = j_{rand}) \\ &= x_{j,i,G} \qquad \text{otherwise}, \end{aligned} \tag{2.11}$$

where

$rand_{i,j}$[0, 1] is a uniformly distributed random number,

$j_{rand}\epsilon$ [1, 2, ... , D], which ensures that $\vec{U}_{i,G}$ gets at least one component from $\vec{V}_{i,G}$ and is instantiated only once for each vector per generation.

Three possible trial vectors may result from uniformly crossing a mutant/donor vector $\vec{V}_{i,G}$ with the target vector $\vec{X}_{i,G}$. These trial vectors are as follows:

1. $\vec{U}_{i,G} = \vec{V}_{i,G}$ such that both the components of $\vec{U}_{i,G}$ are inherited from $\vec{V}_{i,G}$;

2. $\vec{U}'_{i,G}$, in which the first component (j = 1) comes from $\vec{V}_{i,G}$ and the second one (j = 2) from $\vec{X}_{i,G}$;

3. $\vec{U}''_{i,G}$, in which the first component (j = 1) comes from $\vec{X}_{i,G}$ and the second one (j = 2) from $\vec{V}_{i,G}$.

The possible trial vectors due to uniform crossover are illustrated in Fig. 2.8.

2.4.2.4 Selection

To determine whether the target or the trial vector survives to the next generation or not, the selection process is carried out. This operation assures that the total population size remains a constant. The selection operation is described as follows:

$$\begin{aligned} \vec{X}_{i,G+1} &= \vec{U}_{i,G} \qquad \text{if } f(\vec{U}_{i,G}) \leq f(\vec{X}_{i,G}) \\ &= \vec{X}_{i,G} \qquad \text{if } f(\vec{U}_{i,G}) > f(\vec{X}_{i,G}), \end{aligned} \tag{2.12}$$

where $f(\vec{X})$ is the objective function to be minimized. Hence, the population either gets better (with respect to the minimization of the objective function) or remains the same in fitness status, but never deteriorates.

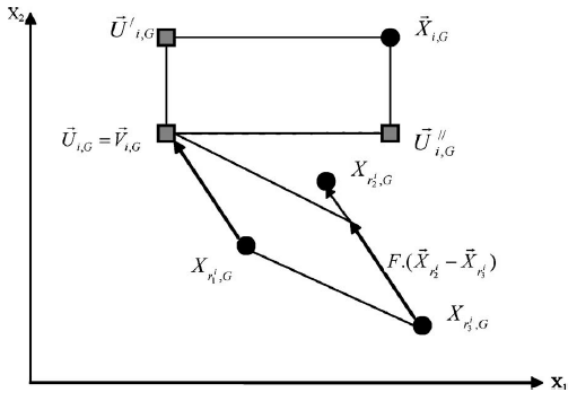

Fig. 2.8: Different possible trial vectors formed due to uniform/binomial crossover between the target and the mutant vectors in 2-D search space.

2.4.2.5 Termination

The above steps are carried out in a loop of iterations. The algorithm terminates when one of the following conditions are satisfied:

1. When the fixed number of generations G_{max} are reached;
2. When the best fitness of the population does not change appreciably over successive iterations;
3. When a pre-specified objective function value has been attained.

2.4.3 Variations of DE

It is the process of mutation that demarcates one DE scheme from another. The mutation scheme in the simple DE discussed in the previous sections, uses a randomly selected vector $\vec{X}_{r1}$ and only one weighted difference vector $F \cdot (\vec{X}_{r2} - \vec{X}_{r3})$ to perturb it. Hence, in literature, this particular mutation scheme is referred to as DE/rand/1. When used in conjunction with binomial crossover, the procedure is called DE/rand/1/bin. The general convention used is DE/x/y/z, where *DE* stands for *differential evolution*, x represents a string denoting the base vector to be perturbed, y is the number of difference vectors considered for perturbation of x, and z stands for the type of crossover being used (exp: exponential; bin: binomial). The other four different mutation schemes, suggested by Storn and Price, are summarized as

$$\text{DE/best/1: } \vec{V}_{i,G} = \vec{X}_{best,G} + F \cdot (\vec{X}_{r^i_{1,G}} - \vec{X}_{r^i_{2,G}}), \tag{2.13}$$

$$\text{DE/target-to-best/1: } \vec{V}_{i,G} = \vec{X}_{i,G} + F \cdot (\vec{X}_{best,G} - \vec{X}_{i,G}) + F \cdot (\vec{X}_{r^i_{1,G}} - \vec{X}_{r^i_{2,G}}), \tag{2.14}$$

$$\text{DE/best/2: } \vec{V}_{i,G} = \vec{X}_{best,G} + F \cdot (\vec{X}_{r^i_{1,G}} - \vec{X}_{r^i_{2,G}}) + F \cdot (\vec{X}_{r^i_{3,G}} - \vec{X}_{r^i_{4,G}}), \tag{2.15}$$

$$\text{DE/rand/2: } \vec{V}_{i,G} = \vec{X}_{r^i_{1,G}} + F \cdot (\vec{X}_{r^i_{2,G}} - \vec{X}_{r^i_{3,G}}) + F \cdot (\vec{X}_{r^i_{4,G}} - \vec{X}_{r^i_{5,G}}). \tag{2.16}$$

The indices r^i_1, r^i_2, r^i_3, r^i_4, and r^i_5 are mutually exclusive integers randomly chosen from the range [1, NP], and all are different from the base index i. These indices are

randomly generated once for each donor vector. $\vec{X}_{best,G}$ is the best individual vector with the best fitness in the population at generation G. Storn and Price suggested a total of ten different working strategies for DE and some guidelines in applying these strategies to any given problem. These strategies were derived from the five different DE mutation schemes outlined above. Each mutation strategy was combined with either the *exponential* type crossover or the *binomial* type crossover. This yielded a total of $52 = 10$ DE strategies.

One drawback that could be observed in all the mutation schemes of the DE is that the formulae show that the convergence power of the DE algorithm is very high, but the exploration is not that good. The difference vectors produced could easily guide all the particles in the population pool to minima. Now these minima could easily be other local minima, rather than the singular global minimum where all the particles should eventually converge at. To overcome this drawback, a new mutation scheme has been proposed as follows:

$$\vec{V}_{i,G} = \vec{X}_{r_1^i,G} + F \cdot (\vec{X}_{r_2^i,G} - \vec{X}_{r_3^i,G}) + \mu \cdot 2 \cdot (rand() - \lambda). \tag{2.17}$$

The DE/rand/1 scheme has been chosen above for this demonstration. λ is a control parameter that defines the range of the randomness generated by $rand()$ function, which is nothing but a random number generator function that picks a number from the standard uniform distribution in the open interval (0, 1). μ is another control parameter that defines the fraction of randomness selected. It lies in the interval [0, 1]. When μ is 0, the random movement is shut off completely. When it is 1, the random movement is maximum. λ is usually a small number, typically close to 0.5. Hence, as a result of this introduction of the random movement parameter in the mutation schemes, there is a very low probability that the particles get stuck at any local minima in the function. Let us call it DE/move/1 for the sake of convenience.

Chapter 3

Comparison of the Algorithms on Standard Test Functions

3.1 Overview

Now we are ready to test the algorithms against some standard test functions. In order to get the best comparison of the results, the algorithms must be initialized with the same parameters. The test function used here include several traditional scenarios, from a simple function with a single minimum to one having a considerable number of local minima of very similar values. They are all continuous or semi-continuous functions. The algorithms have been benchmarked against six standard test functions that are more or less difficult to deal with [1]. In each case, the known minimal value is zero and one wishes to reach it with high accuracy and in the shortest number of generations possible. Also, the algorithm itself should not take a long run time to execute. These three parameters are the main features that earn victory points for the algorithms.

The RGA, PSO, DE/rand/1/bin, and DE/move/1/bin algorithms are uniformly configured. After many runs of all the algorithms, it was found out that, to get a good idea of the big picture of whats happening, around 10,000 generations were necessary for each of them. The slower algorithms which do much better in convergence could have been easily truncated to a lot fewer generation runs. Similarly, the algorithms which perform poorly on convergence would definitely have converged to lower values, provided they were run over longer generations. But a standard protocol was maintained to compare them on the same pedestal. Hence, the number of generations was fixed to 10,000 in every case, and the convergence plots were studied. We might have as well fixed the run-time of the algorithms to a common value, but since the run-times of the different algorithms are almost an order of magnitude apart from each other, it was not advisable to do so. Also, the population sizes for all the algorithms were fixed at 20. The PSO requires about 20 - 40 particles for effective

swarming. The RGA was fixed at 40 members, because in every iteration, half the population is discarded. The DE converges at far lesser number of particles. The convergence plots give a good idea on the following three main characteristics of these algorithms:

1. Run-time of the algorithm,
2. Number of generations taken to converge (when the cost trickles down to small variations over further generations or does not change appreciably),
3. Fitness value.

Every algorithm was run on the six functions, five times each, and the average values of the above three parameters were calculated for each case. Every table in the following sections shows the three parameters obtained through the experiments for the five cases of each algorithm, and also the average values calculated are shown in the last row. RT stands for run time in seconds, CP stands for convergence point in number of generations, and FV stands for fitness value.

3.1.1 Tripod Function

The tripod function is given by the formula

$$f(x_1, x_2) = p(x_2)(1 + p(x_1)) + |x_1 + 50p(x_2)(1 - 2p(x_1))| + |x_2 + 50(1 - 2p(x_2))|, \quad (3.1)$$

with $p(u) = 1$ if $u \geq 0$, $p(u) = 0$ if $u < 0$.

This function is very fast to compute and theoretically easy, but hard to optimize the real minimum 0. This problem misleads many algorithms, which are easily trapped in one or other of the local minima. The real minimum 0 is at the point (0, -50). This function has just two variables or dimensions to optimize. Every variable in the test are set to have a range of [-100, 100]. Figure 3.1 shows a typical tripod function. Table 3.1 shows the convergence results for all the algorithms on the tripod function. Figure 3.2 shows the convergence plots.

We can see that the PSO took the least time to run 10,000 generations, but the DE/rand/1/bin was the fastest to converge with about just 119 generations, and an extremely good fitness value of almost 0. This algorithm did get caught in the local minimum at (50, -50), but far fewer times than RGA or PSO. In this case, the addition of random movement in the DE did not do a great job.

3.1.2 Alpine Function

The alpine function is given by the formula

$$f(x_d) = \sum_{d=1}^{D} |x_d sin(x_d) + 0.1x_d|, \tag{3.2}$$

with $d = 1, 2, ..., 10$

This function has many local and global minima of zero value. The surface of space is not completely symmetrical compared to the origin. One can imagine the plot to have the French Côte d'Azur on the south and the Mont Blanc as the highest summit. Every

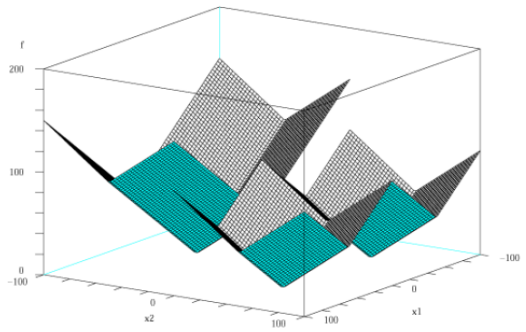

Fig. 3.1: Tripod function.

Table 3.1: Convergence results of the tripod function.

Case	RGA			PSO			DE/rand/1/bin			DE/move/1/bin		
	RT	CP	FV	RT	CP	FV	RT	CP	FV	RT	CP	FV
1	27.6601	9979	0.6622	10.1182	3082	1.0561	76.3406	48	0	61.2696	53	1.0001
2	23.1320	7068	2.0528	7.2610	4619	0.0309	56.6730	262	0.0002	58.7845	493	2.0001
3	31.2813	2756	1.6995	10.5586	9500	1.0600	76.8792	131	0	70.3031	1589	1
4	24.3613	2267	1.1284	9.3369	5862	0.0261	68.5777	56	0	85.3571	2921	1.0001
5	32.2956	4085	0.2499	9.7625	5211	1.0184	67.0837	98	0	69.6330	144	1
Average	27.7461	5231	1.1586	9.4074	5654.8	0.6383	69.1108	119	4e-5	69.0695	1240	1.2001

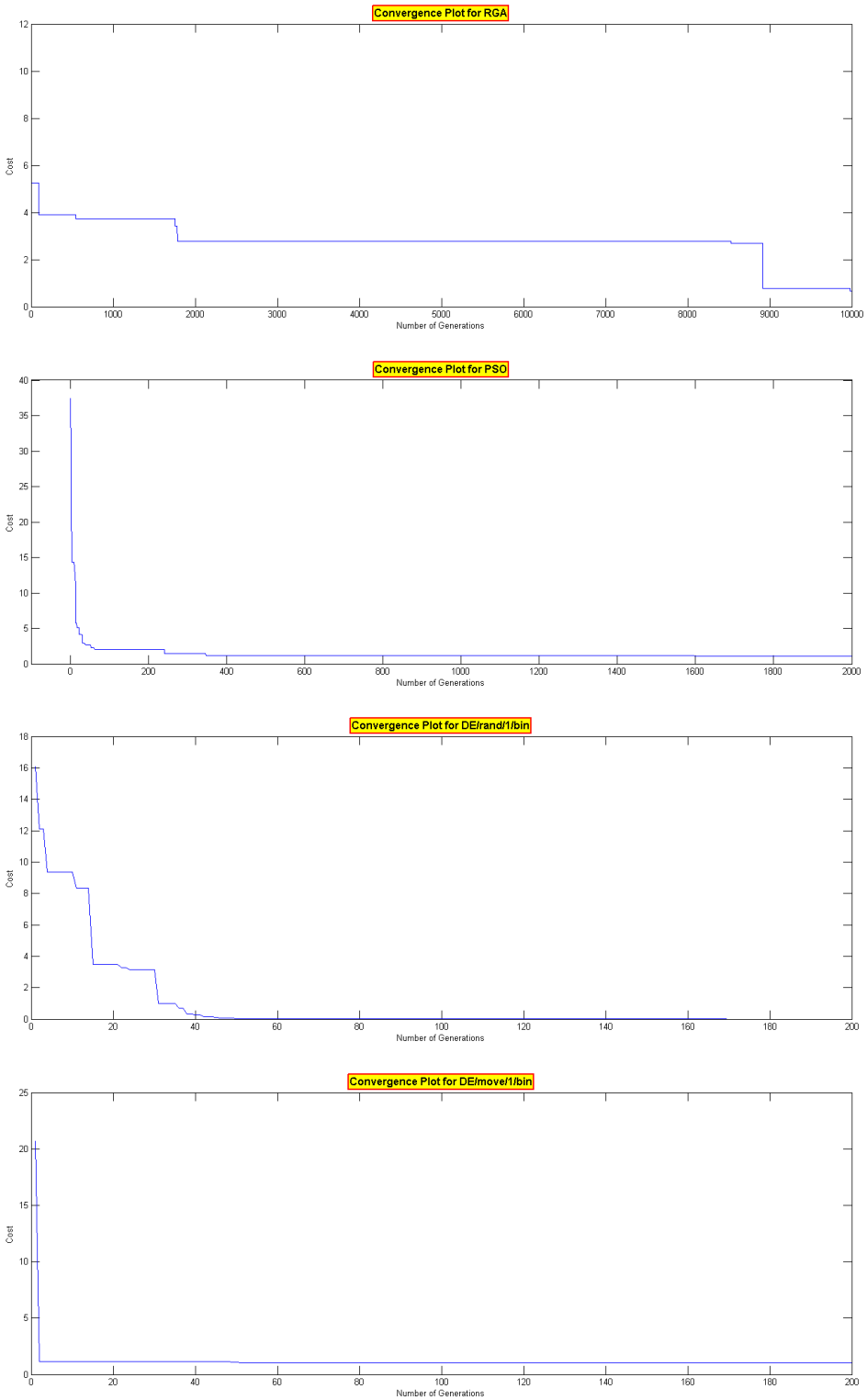

Fig. 3.2: Convergence plots of the tripod function.

variable in the test are set to have a range of [-10, 10]. Figure 3.3 shows a typical alpine function. Table 3.2 shows the convergence results for all the algorithms on the alpine function. Figure 3.4 shows the convergence plots.

As we can see, the DE/rand/1/bin algorithm far out-performed all its contemporaries. It converged much faster than the others, at about 333 generations, and with a perfect fitness value of 0. But again, the run time of this algorithm was really large. A notable similarity in terms of the fitness values, is that the performance of the PSO and the DE/move/1/bin was almost the same. The PSO algorithm runs much faster than the DE/move/1/bin algorithm, but the convergence point of the latter is faster than the former. So it is a trade-off between the two factors, and it is left to the user to choose his/her suitable algorithm for the problem under consideration.

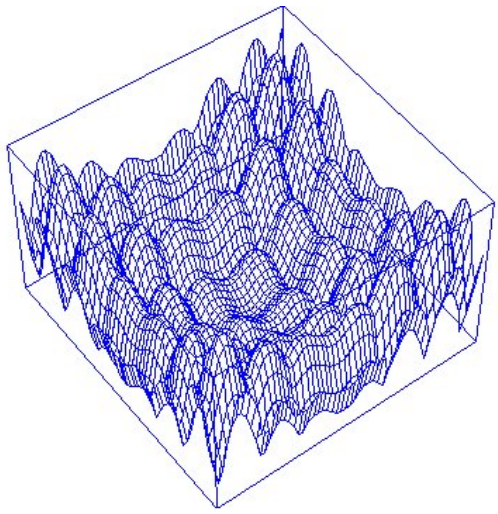

Fig. 3.3: Alpine function.

Table 3.2: Convergence results of the alpine function.

Case	RGA			PSO			DE/rand/1/bin			DE/move/1/bin		
	RT	CP	FV	RT	CP	FV	RT	CP	FV	RT	CP	FV
1	80.9463	9866	1.8916	46.8412	5041	0.0055	270.1911	428	0	344.4248	3464	0.0100
2	56.0135	9542	1.1020	31.0046	1024	1.0695e-139	311.1308	442	0	320.6588	3932	0.0090
3	60.9471	2152	2.3453	31.3519	3650	4.5815e-8	317.3946	242	0	322.6788	1632	0.0069
4	62.4662	2053	1.2871	38.6755	2455	2.2424e-9	283.2997	399	0	313.2167	404	0.0091
5	59.4170	7526	2.0729	32.9618	1239	5.0792e-14	312.0844	158	0	274.6054	304	0.0094
Average	63.9580	6227.8	1.7398	36.1670	2681.8	0.0011	298.8201	333.8	0	315.1169	1947.2	0.0089

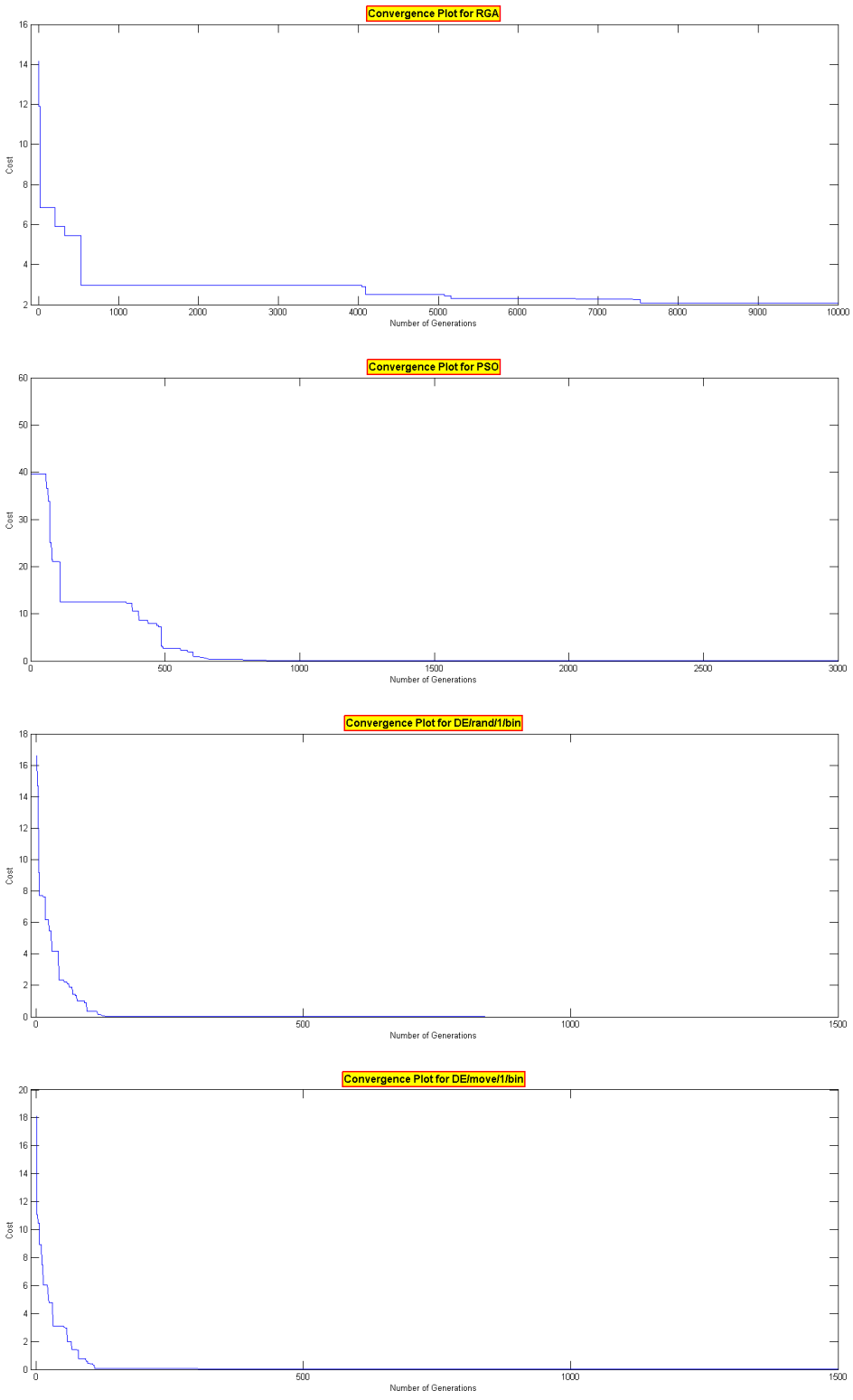

Fig. 3.4: Convergence plots of the alpine function.

3.1.3 Parabola Function

The parabola function is given by the formula

$$f(x_d) = \sum_{d=1}^{D} x_d^2, \tag{3.3}$$

with $d = 1, 2, ..., 30$

This function has only one minimum. In two dimensions, this function is a paraboloid, but it sometimes refered to a "sphere" because of its equation. Parabolas can open up, down, left, right, or in some other arbitrary direction. Any parabola can be repositioned and rescaled to fit exactly on any other parabola, i.e., all parabolas are similar. Every variable in the test are set to have a range of [-20, 20]. Figure 3.5 shows a typical parabola function. Table 3.3 shows the convergence results for all the algorithms on the parabola function. Figure 3.6 shows the convergence plots.

We notice that the PSO fails entirely to converge in this case. The reason could be because of the high stochastic nature of the algorithm. The RGA did not do great as well, though it can be predicted that it would have eventually converged, provided that it ran for a much longer time. Once again, the DE/rand/1/bin out-performed all the other

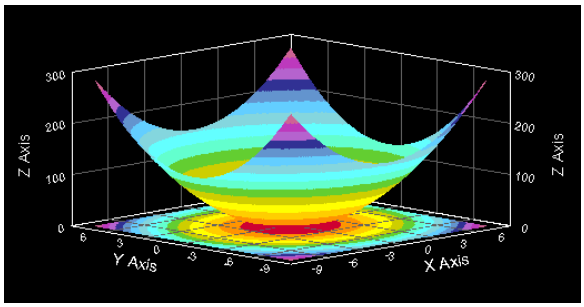

Fig. 3.5: Parabola function.

Table 3.3: Convergence results of the parabola function.

Case	RGA			PSO			DE/rand/1/bin			DE/move/1/bin		
	RT	CP	FV	RT	CP	FV	RT	CP	FV	RT	CP	FV
1	116.0121	9428	217.3472	61.7784	10000	4.1985e4	896.4982	233	6.3731e-155	821.5740	339	0.4897
2	167.8310	9599	179.6442	108.6506	10000	2.6754e4	1018.3252	180	1.2725e-155	903.8873	348	2.3932
3	170.0253	9830	235.2782	106.3661	10000	3.1525e4	990.5752	215	9.1639e-157	1023.6962	323	0.4430
4	123.6254	9909	262.6918	67.9691	10000	3.8738e4	907.3945	247	2.8155e-156	937.7781	246	0.3663
5	153.8074	9003	262.1658	93.2714	10000	3.7045e4	786.2998	205	2.0411e-154	915.1588	252	0.3851
Average	146.2602	9553.8	231.4254	87.6071	10000	3.5209e4	919.8186	216	5.6860e-155	920.4189	301.6	0.8155

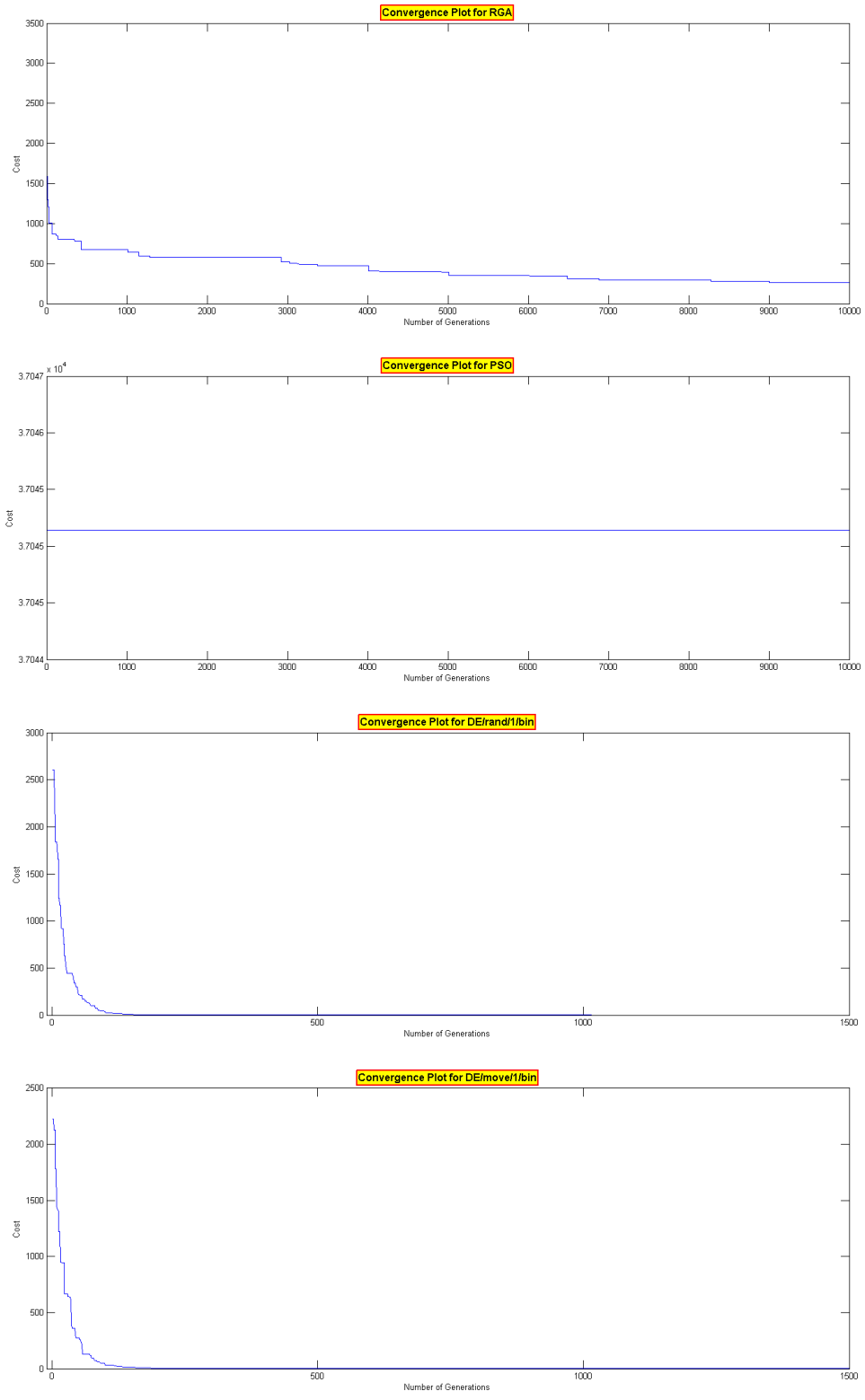

Fig. 3.6: Convergence plots of the parabola function.

algorithms, it converged to almost 0 at a very early stage of about 216 generations.

3.1.4 Griewank Function

The griewank function is given by the formula

$$f(x_d) = \frac{\sum_{d=1}^{D}(x_d - 100)^2}{4000} - \prod_{d=1}^{D} cos\left(\frac{x_d - 100}{\sqrt{d}}\right) + 1, \tag{3.4}$$

with $d = 1, 2, ..., 30$

This function is relatively more difficult. The global minimum 0 is at (100, 100) and is almost indistinguishable from many closely packed local minima that surround it. On the one hand, that tends to increase the difficulty of the problem, but, on the other hand, because the local minima are very close together, it is rather easy to escape from them. Every variable in the test are set to have a range of [-300, 300]. Figure 3.7 shows a typical griewank function. Table 3.4 shows the convergence results for all the algorithms on the griewank function. Figure 3.8 shows the convergence plots.

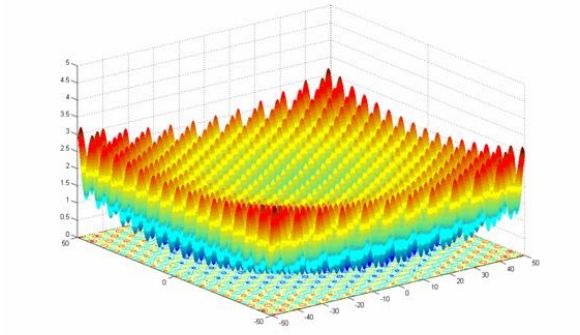

Fig. 3.7: Griewank function.

Table 3.4: Convergence results of the griewank function.

Case	RGA			PSO			DE/rand/1/bin			DE/move/1/bin		
	RT	CP	FV	RT	CP	FV	RT	CP	FV	RT	CP	FV
1	134.6176	8608	16.5516	140.8024	10000	1.9127e3	564.6294	390	0	567.6047	524	0.0161
2	123.4726	9772	14.1644	59.6755	10000	2.4183e3	560.9375	538	0	562.2409	396	0.0171
3	126.2409	7820	14.7770	67.4313	10000	1.8345e3	576.7580	530	0	552.9202	553	0.0194
4	114.3179	7108	15.3040	63.8457	9205	2.5210	562.9533	324	0	550.8040	592	0.0193
5	133.4105	7281	14.0451	57.7485	10000	2.2105e3	560.9396	560	0	553.3350	597	0.0205
Average	126.4119	8117.8	14.9684	77.9007	9841	2.1794e3	565.2436	468.4	0	557.3810	532.4	0.0185

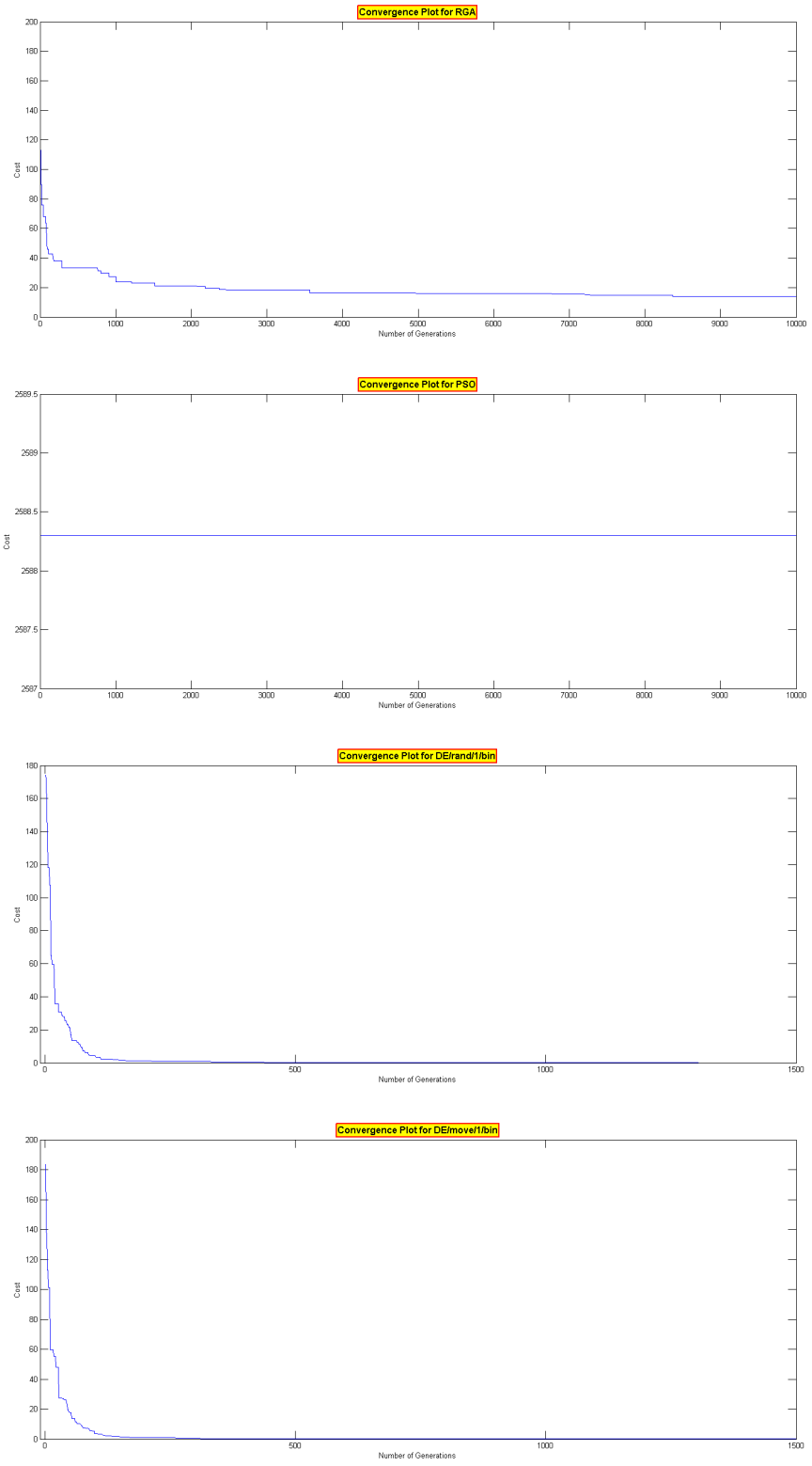

Fig. 3.8: Convergence plots of the griewank function.

Once again, the PSO fails to converge in this case. The RGA needs a really long time to do so. The DE/move/1/bin, though slower than the RGA, performed much better at convergence. The DE/rand/1/bin converged to a perfect 0 in just about 468 generations.

3.1.5 Rosenbrock Function

The rosenbrock function is given by the formula

$$f(x_d) = \sum_{d=1}^{D-1} (1 - x_d)^2 + 100(x_d^2 - x_{d+1})^2, \tag{3.5}$$

with $d = 1, 2, ..., 30$

There is a barely noticeable global minimum at (1, 1) in this function. It is inside a long, narrow, parabolic shaped flat valley. To find the valley is trivial. To converge to the global minimum, however, is difficult for the majority of optimization algorithms. Every variable in the test are set to have a range of [-10, 10]. Figure 3.9 shows a typical rosenbrock function. Table 3.5 shows the convergence results for all the algorithms on the rosenbrock function. Figure 3.10 shows the convergence plots.

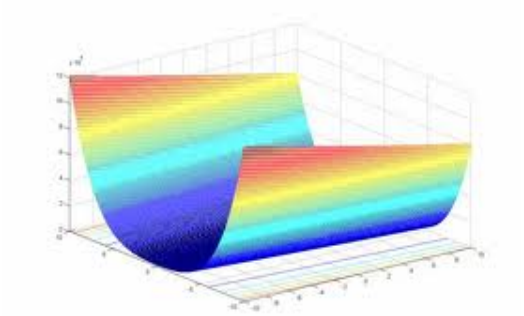

Fig. 3.9: Rosenbrock function.

Table 3.5: Convergence results of the rosenbrock function.

Case	RGA			PSO			DE/rand/1/bin			DE/move/1/bin		
	RT	CP	FV	RT	CP	FV	RT	CP	FV	RT	CP	FV
1	225.2946	9663	7.1074e4	93.5374	10000	6.4716e8	563.6867	230	26.6747	552.4418	242	71.4514
2	221.0375	6443	5.8801e4	100.4742	10000	7.0100e8	563.1304	271	27.6888	582.9001	506	64.0891
3	231.2592	8783	4.9946e4	93.9829	10000	6.8449e8	545.1125	321	26.3643	567.8060	304	77.3858
4	111.0676	9708	6.4483e4	83.4870	10000	6.9977e8	542.0625	225	29.1996	565.1076	472	73.2780
5	114.1827	8567	4.4431e4	51.7515	10000	8.5225e8	499.4760	299	29.3288	519.5395	410	75.0741
Average	180.5683	8632.8	5.7747e4	84.6466	10000	7.1693e8	542.6936	269.2	27.8512	557.5590	386.8	72.2557

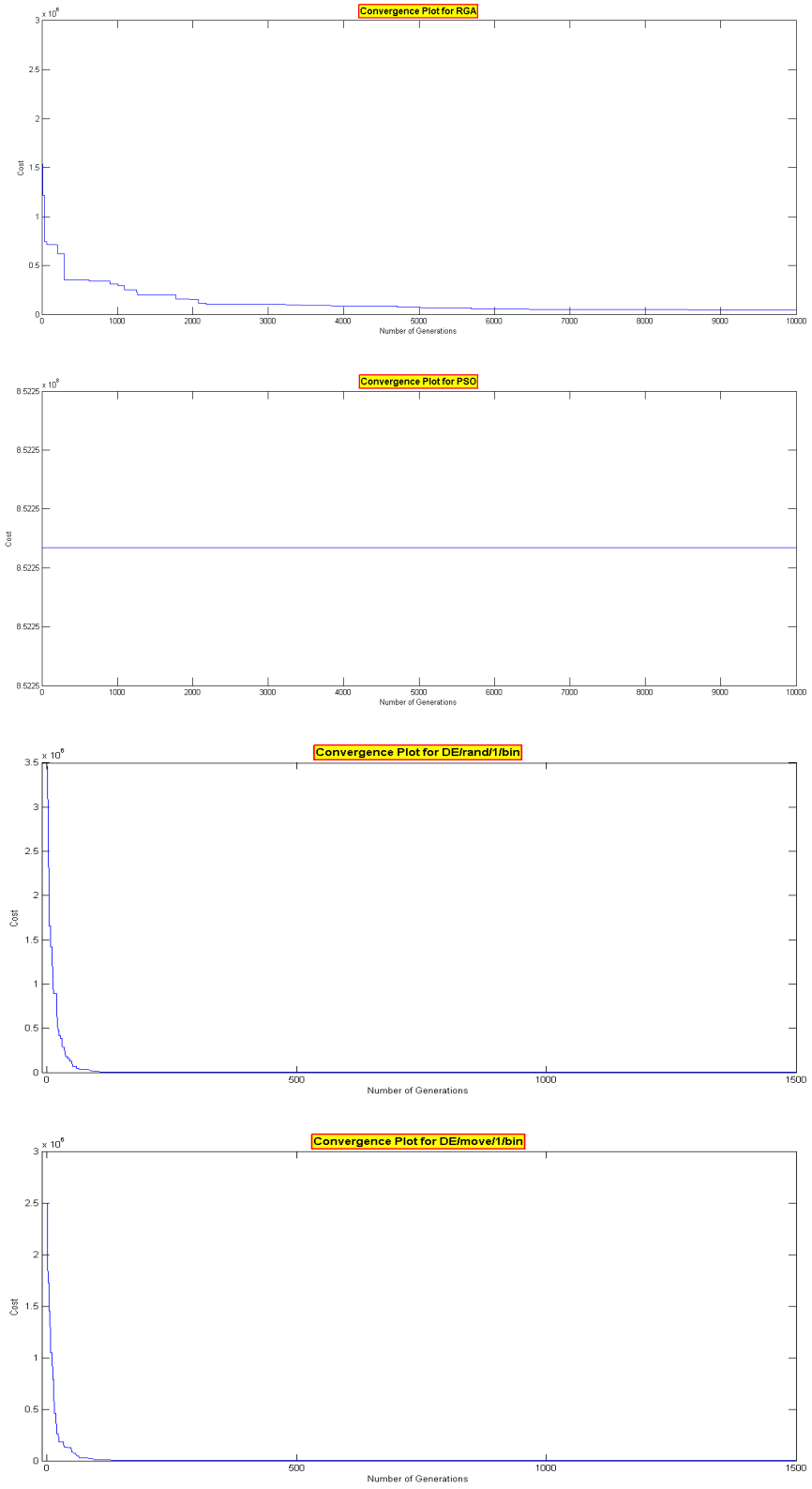

Fig. 3.10: Convergence plots of the rosenbrock function.

As expected, the rosenbrock function proved to be hard for all the algorithms. The RGA and PSO failed completely, though the RGA was trying to trickle down towards convergence. The DE algorithms, however, did fairly well in comparison to the other two. The DE/rand/1/bin in particular, quickly converged to a fitness value of 28 in about 269 generations. This is a good sign which indicates that the algorithm could be modified to easily perform better at convergence.

3.1.6 Ackley Function

The ackley function is given by the formula

$$f(x_d) = -20e^{-0.2\sqrt{\frac{\sum_{d=1}^{D} x_d^2}{D}}} - e^{\frac{\sum_{d=1}^{D} cos(2\pi x_d)}{D}} + 20 + e, \tag{3.6}$$

with $d = 1, 2, ..., 30$

The ackley function apparently resembles the alpine function, but is actually more difficult, even with the same dimensionality. The "basin of attraction" of the global minimum is narrower, which decreases the effectiveness of random displacements. This function is an n-dimensional highly multimodal function that has a large number of local minima but only one global minimum. The function has a global minimum at the origin (0, 0) with value 0. Every variable in the test are set to have a range of [-30, 30]. Figure 3.11 shows a typical ackley function. Table 3.6 shows the convergence results for all the algorithms on the ackley function. Figure 3.12 shows the convergence plots.

As can be seen, the RGA and PSO seem to be showing a slow convergence, and it would take really long run times for them to probably achieve good convergence at the global minimum with small error margins. The DE/rand/1/bin algorithm converged to almost 0 at about 530 generations.

3.2 Results

From the tests conducted, we can clearly see that the DE/rand/1/bin showed the best convergence performance on all the functions. It takes far lesser number of generations

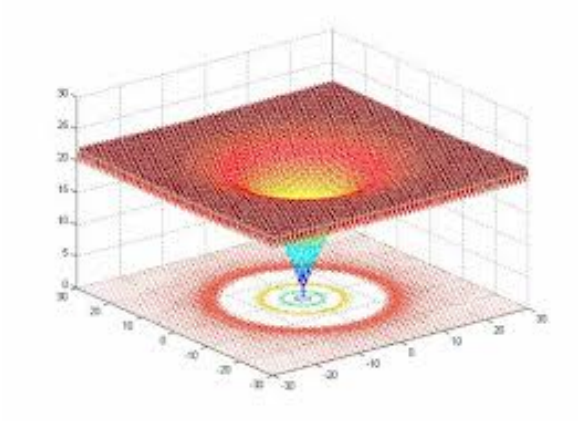

Fig. 3.11: Ackley function.

Table 3.6: Convergence results of the ackley function.

Case	RGA			PSO			DE/rand/1/bin			DE/move/1/bin		
	RT	CP	FV	RT	CP	FV	RT	CP	FV	RT	CP	FV
1	138.1576	9082	13.2499	66.7518	8110	20.8553	904.4990	556	4.4409e-15	913.1812	6474	1.0484
2	139.8894	9882	13.2319	75.8110	4963	20.7908	900.9805	507	4.4409e-15	905.4703	4493	1.1212
3	143.3215	9642	13.0235	60.2047	3926	20.8478	887.4354	550	4.4409e-15	892.0827	7790	1.0888
4	117.8260	9229	12.3351	82.1797	2802	20.7657	882.7675	529	4.4409e-15	875.6078	8492	0.9637
5	117.0406	9091	13.7425	85.9682	9959	20.7792	924.1237	509	4.4409e-15	939.9106	7957	1.0155
Average	131.2470	9385.2	13.1166	74.1831	5952	20.8078	899.9612	530.2	4.4409e-15	905.2505	7041.2	1.0475

to converge to the minimum value, and with the least error margin as well. The major drawback of this algorithm is that it has a relatively high run time, or it takes a relatively longer time to compute any particular problem with the same initial conditions. These initial conditions are the number of generations, the population size, the dimensions, etc.

The PSO and the RGA have lower run times, but the convergence performance was not very appreciable. The only reason the RGA ran slower than the PSO is probably because the population size in every problem was doubled for the RGA, for reasons previously discussed. The DE/move/1/bin algorithm showed good convergence for some problems, but not so good for the others. Hence, the DE/rand/1/bin algorithm was chosen and applied to certain real world problems, which shall be discussed more in detail in the following chapters. Henceforth, the DE/rand/1/bin algorithm shall be referred to as just the DE algorithm.

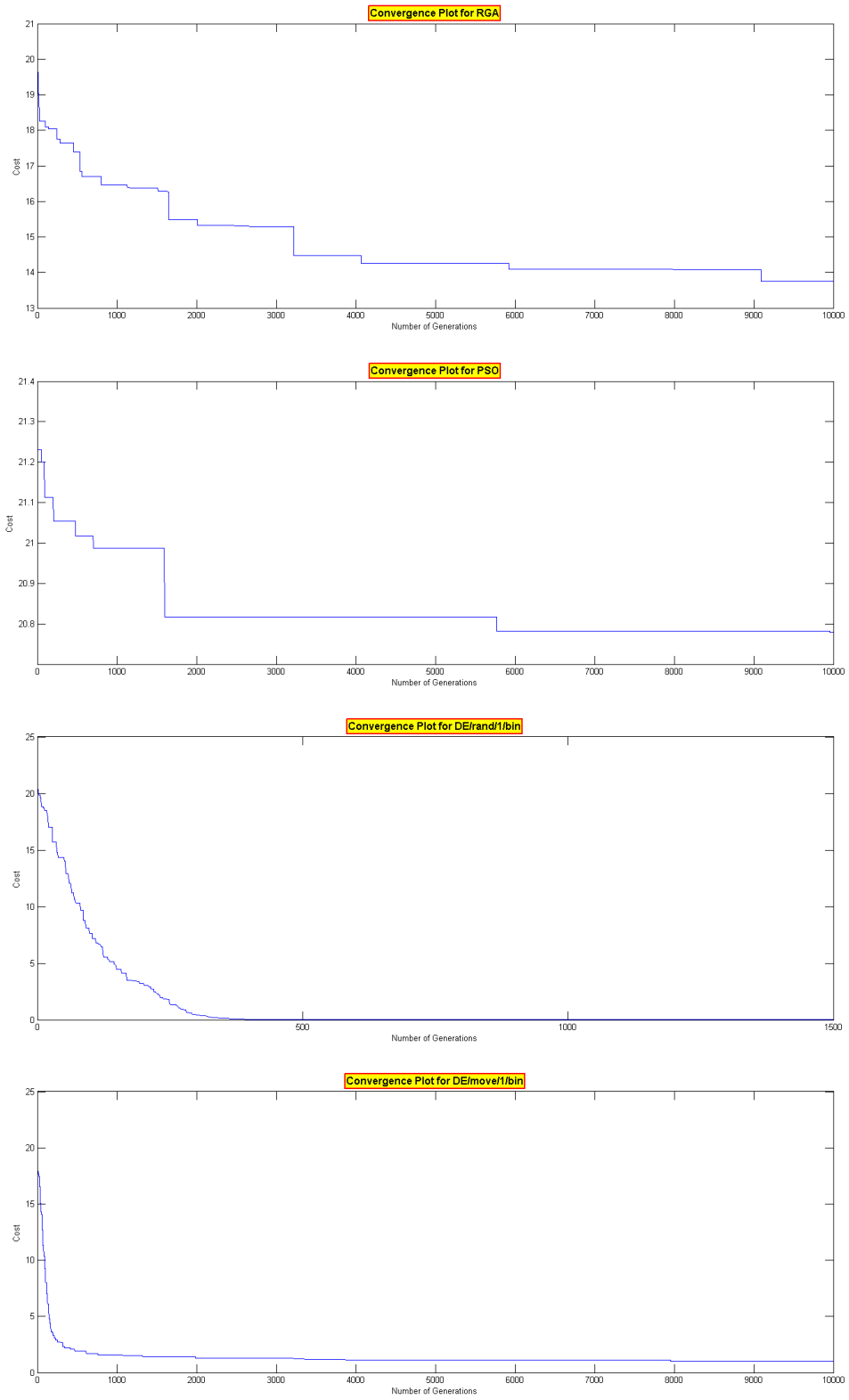

Fig. 3.12: Convergence plots of the ackley function.

Chapter 4

Auroral Oval Boundaries Model

4.1 Introduction

On Earth, the impact zones of energetic particles from the Sun, i.e., a circular belt of auroral emissions around each geomagnetic pole, are known as the auroral ovals [14]. The Sun emits a continuous stream of charged particles called the solar wind. This wind interacts with the magnetic field of the Earth and produces large electrical currents. These currents flow along the magnetic field lines down into the upper atmosphere surrounding the north and south magnetic poles. The ionization of the atomic gases due the collision of the neutral gas atoms with the precipitated charged particles cause the atmospheric gases to glow like the gas in a fluorescent tube. A very large quantity of energy is deposited in the upper atmosphere during an auroral display. Figure 4.1 shows the interaction of the solar wind with the Earth's magnetosphere. Figure 4.2 shows a view of the entire auroral oval taken by the Dynamics Explorer 1 satellite on 8th November, 1981 from high above the north polar region. This image was taken at an altitude of approximately 20,000 km. Here the glowing oval of auroral emission is about 4,500 km in diameter.

The location and size of these ovals have been studied extensively in the second half of the 20th century. During the International Geophysical Year (1957−1958), auroral occurrence was determined from all-sky camera studies for a wide variety of activity levels. The resulting figure was made up from a statistical study showing the pole-ward and equator-ward auroral boundaries of the 75% occurrence probability [15–17]. The relationship between the morphology of the auroral oval and the level of geomagnetic activity allows the development of models of the location of the aurora, independent of the vagaries of auroral observations [18, 19].

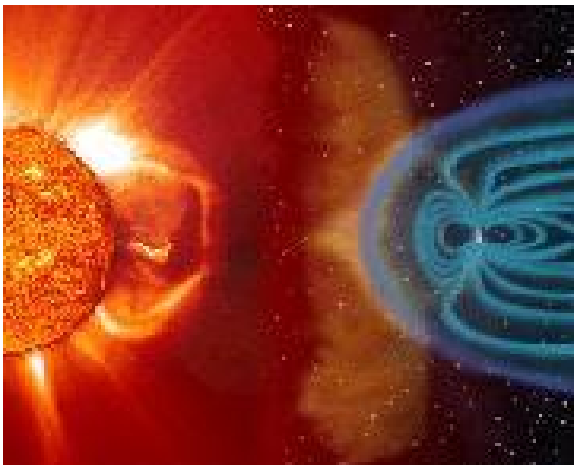

Fig. 4.1: Interaction between the solar wind and the Earth's magnetosphere.

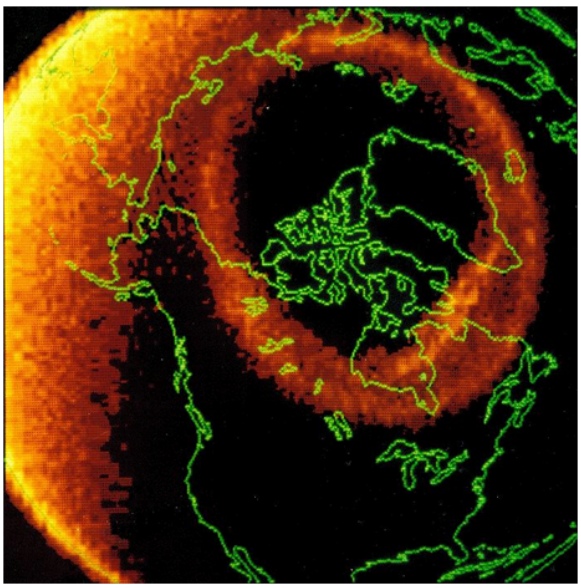

Fig. 4.2: Auroral oval image taken by the Dynamics Explorer 1 satellite over the north polar region on 8th November, 1981.

Over the last three decades, new models have evolved that use data of particle precipitation measured by polar orbiting satellites [20–23]. Recently, Zhang and Paxton [24] developed an auroral oval model based on data from the Ultraviolet Imager (GUVI) on board the TIMED (Thermosphere Ionosphere Mesosphere Energetics and Dynamics) satellite. The planetary geomagnetic activity index Kp [25] is directly related to the size and location of the auroral oval, and it is used as an input parameter to the above models.

The advent of stationary spacecraft between the Sun and the Earth has led to studies of the relationship between the structure of the solar wind and the resulting auroral and geomagnetic disturbances. When the input data come from a satellite located approximately 1 h upstream in the solar wind, the resultant *predicted Kp* forecast is relatively short term [26, 27]. It gives a 1- or 4-h warning. This can be very useful both for auroral observers and experiments that are dependent on the location and intensity of the aurora.

In this chapter, we are going to discuss the Feldstein-Starkov model. It would be followed by a brief description about the British Antarctic Survey (BAS) data and the Auroral Electrojet (AE) indices, and the way they are formatted. A new model has been proposed using this data, to calculate the auroral oval boundaries at both the pole-ward and equator-ward ends. The BAS data is used as the standard against which the model has been trained upon, which in turn is driven by the AL index. AL index is one of the AE indices, which we will talk about later in this chapter. The DE algorithm is used to obtain optimal coefficients for the formulae used for these boundaries. This model is then compared against the Feldstein-Starkov model and the results are presented.

4.2 The Feldstein-Starkov Model

AE index is an auroral electrojet index obtained from a number (usually greater than 10) of stations distributed in local time in the latitude region that is typical of the northern hemisphere auroral zone. For each of the stations, the north-south magnetic perturbation H is recorded as a function of universal time. A superposition of these data from all the stations enables a lower bound or maximum negative excursion of the H component to be determined; this is called the AL index. This index describes the polar or planetary

magnetic disturbances that occur during auroras. The range of the index is of the order of ±800 nT. In terms of Kp index it is given as [28]

$$AL = c_0 + c_1 \cdot K_p + c_2 \cdot K_p^2 + c_3 \cdot K_p^3. \tag{4.1}$$

Table 4.1 shows the coefficients c_i for i ϵ [0, . . ., 3].

The value of the Kp index varies from 0 to 9 with 0 being very quiet and 5 or more indicating geomagnetic storm conditions. In detail, Kp represents a 3-h weighted average from a global network of magnetometers measuring the maximum deviation in the horizontal component of the Earth's magnetic field. In coordinates of corrected geomagnetic colatitude, θ, the boundaries of the oval are expressed as

$$\theta_m = A_{0m} + A_{1m}cos[15(t + \alpha_{1m})] + A_{2m}cos[15(2t + \alpha_{2m})] + A_{3m}cos[15(3t + \alpha_{3m})], \tag{4.2}$$

where A_{im} and α_{im} for i ϵ [0, . . ., 3] are amplitudes in units of degrees of latitude and phases in units of decimal hours, respectively. t is local time. m represents the modes for different boundaries, i.e., the pole-ward (m = 0), the equator-ward (m = 1), and the diffuse aurora boundaries (m = 2). Starkov [19] uses a new third-order polynomial for both the A_{im} and α_{im} coefficients, as shown

$$A_{im} \text{ or } \alpha_{im} = b_{0m} + b_{1m}log_{10}|AL| + b_{2m}log_{10}^2|AL| + b_{3m}log_{10}^3|AL|. \tag{4.3}$$

Figure 4.3 shows all the above coefficients for the different modes. The coefficients c_i and b_{im} for i ϵ [0, . . ., 3] obtained by Starkov are presented in his work [19]. Equation (4.1) is used to get AL values. Equation (4.2) is used to obtain the boundaries using the coefficients calculated by using equation (4.3). Figure 4.4 is a representation of the

Table 4.1: Coefficients to convert Kp to AL indices.

c_0	c_1	c_2	c_3
18	-12.3	27.2	-2

Units	[°]	[°]	[°]	[°]	[h]	[h]	[h]
	A_{00}	A_{10}	A_{20}	A_{30}	α_{10}	α_{20}	α_{30}
m = 0 Poleward boundary of the auroral oval							
b_{00}	−0.07	−10.06	−4.44	−3.77	−6.61	6.37	−4.48
b_{10}	24.54	19.83	7.47	7.90	10.17	−1.10	10.16
b_{20}	−12.53	−9.33	−3.01	−4.73	−5.80	0.34	−5.87
b_{30}	2.15	1.24	0.25	0.91	1.19	−0.38	0.98
	A_{01}	A_{11}	A_{21}	A_{31}	α_{11}	α_{21}	α_{31}
m = 1 Equatorward boundary of the auroral oval							
b_{01}	1.61	−9.59	−12.07	−6.56	−2.22	−23.98	−20.07
b_{11}	23.21	17.78	17.49	11.44	1.50	42.79	36.67
b_{21}	−10.97	−7.20	−7.96	−6.73	−0.58	−26.96	−24.20
b_{31}	2.03	0.96	1.15	1.31	0.08	5.56	5.11
	A_{02}	A_{12}	A_{22}	A_{32}	α_{12}	α_{22}	α_{32}
m = 2 Equatorward boundary of the diffuse aurora							
b_{02}	3.44	−2.41	−0.74	−2.12	−1.68	8.69	8.61
b_{12}	29.77	7.89	3.94	3.24	−2.48	−20.73	−5.34
b_{22}	−16.38	−4.32	−3.09	−1.67	1.58	13.03	−1.36
b_{32}	3.35	0.87	0.72	0.31	−0.28	−2.14	0.76

Fig. 4.3: Feldstein-Starkov expansion coefficients for the auroral boundaries.

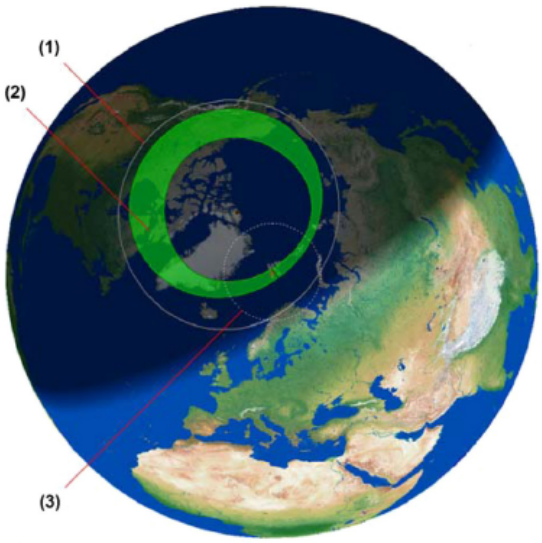

Fig. 4.4: Feldstein-Starkov oval with K_p index of 3. (1) Equator-ward boundary of the diffuse aurora, (2) Feldstein-Starkov aurora oval, (3) Field of view aurora observer.

Feldstein-Starkov oval mapped onto the Earth during a K_p index of 3, projected by the SvalTrackII software.

4.3 British Antarctic Survey Data

The auroral oval boundary locations derived from the IMAGE FUV images, were downloaded for the storm period from the British Antarctic Survey (BAS) group, which is a

component of the Natural Environment Research Council (NERC) based in Cambridge, United Kingdom. The boundaries derived from the Wideband Imaging Camera (WIC) detector of the FUV instrument were used because the detector images the aurora in broad band for maximum spatial resolution day and night. The FUV instrument has two other dtectors. The Spectrographic Imager (SI) measures different types of aurora and separates them by wavelength and measures proton induced aurora by removing the bright geocorona emissions. The Geocorona photometers (GEO) observe the distribution of the geocorona emissions and derive the magnetospheric hydrogen content responsible for neutral atom generation in the magnetosphere.

BAS has data available from May 2000 until October 2002. The data represents 48 points for every time stamp. The first 24 points are the auroral boundary locations for the equatorward boundary, and the next 24 points are the same for the poleward boundary. These locations are given for 1 hour magnetic local time (MLT) bins in ascending order from 00:30 to 23:30 MLT. The first bin is comprised of 0 to 1 MLT, the next one of 1 to 2 MLT, and so on. The locations which could not be established successfully are reported with NaN values. The boundary data file format is shown in Table 4.2. The methodology to estimate the poleward and equatorward boundaries of auroral emissions is enumerated as follows.

1. *IMAGE FUV image* - The IMAGE FUV team provides the IMAGE data in the NASA universal data format (UDF). The FUVIEW3 software is used to process the IMAGE data. It makes use of UDF software to convert the data into IDL format. The final resultant contains information like the original auroral image, the magnetic latitude, the magnetic longitude, and the local time of every pixel of the image in the APEX coordinate system.

2. *Coversion of the image to magnetic coordinates* - Every IMAGE FUV auroral image is converted into the Altitude Adjustment Corrected Geomagnetic (AACGM) system. Figure 4.5 shows an auroral image in AACGM coordinates taken by the WIC instrument between 00 and 06 UT on 1st February, 2001.

Table 4.2: Auroral oval boundary data file format.

Column Number	Data
1	Timestamp in UT in YYYY-MM-DD HH:MM:SS format
2	IMAGE FUV Instrument Code
3 to 26	Magnetic latitude of equatorward boundary locations in AACGM coordinates
27 to 50	Magnetic latitude of poleward boundary locations in AACGM coordinates

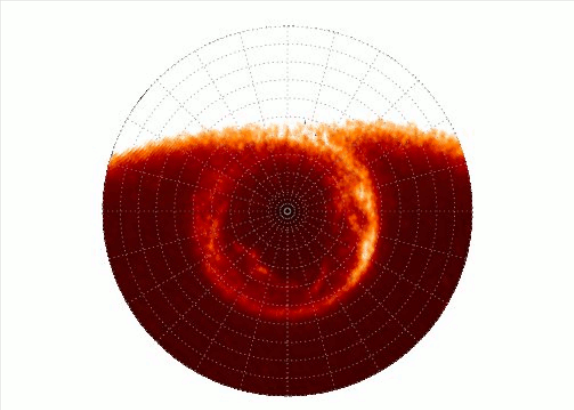

Fig. 4.5: Auroral image on 1st February, 2001 taken by the WIC instrument.

3. *Latitudinal intensity profiles* - The intensity profile for the image is created by first dividing every image into 24 segments. Each segment covers 1 hour of magnetic local time (MLT). The average intensity across bins of 1° magnetic latitude in the range of 50° to 90° AACGM is calculated, and this is used to construct the intensity profile.

4. *Function fits to the intensity profiles* - Auroral emission models are then created by fitting functions to every intensity profile. Two functions are used in this case, one with a single Gaussian component and a quadratic background, and the other with two Gaussian components and a quadratic background. If the emissions occur in a continuous oval, the former function works better. The latter function is more suitable for ovals that show bifurcations. Figure 4.6 shows an example profile with a single Gaussian fit (blue curve) and a double Gaussian fit (red curve). These fits provide estimates of the amplitude, centre, and width of the one or two main peaks in the intensity profile.

5. *Ranking and selection* - The two functions are ranked according to fitness values, and the one with a lower value is selected as the better model for that profile. The cost

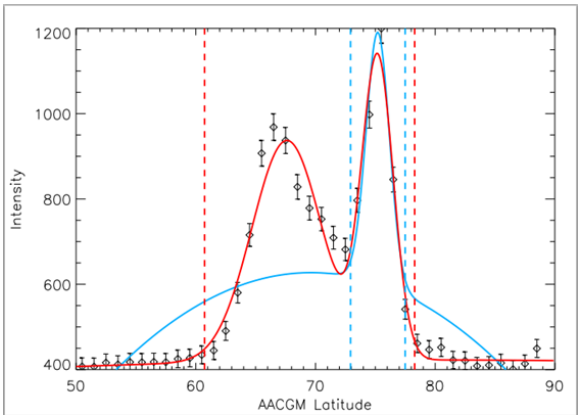

Fig. 4.6: Gaussian function fits to an intensity profile.

function methodology used here is the reduced chi-squared goodness-of-fit statistic method.

6. *Estimation of the auroral boundaries* - The full width at half maximum (FWHM) is used on these intensity profiles to obtain the equatorward and poleward boundaries. If the single Gaussian component is used, the center of the Gaussian peak plus the FWHM gives the poleward boundary and the center of the Gaussian peak minus the FWHM gives the equatorward boundary. When the double Gaussian component function is used, both the peaks are taken into consideration and the offsets and FWHM's from the Gaussian centers are calculated. The most equatorward and poleward offsets are used as the two boundaries.

7. *Elimination of bad boundaries* - Those boundary locations that were not obtained successfully are discarded. These bad locations occur if the amplitude of the fitted Gaussian is less than zero, if the center of the fitted Gaussian does not lie within the latitude range of the intensity profile, if the amplitude of the fitted Gaussian does not exceed the amplitude of the background by at least 10% of the background intensity, if the amplitude of the secondary peak of the double Gaussian function is not at least 20% of the amplitude of the main peak, if the width of the fitted Gaussian exceeds the total range of the intensity profile or does not exceed 1°, the reduced chi-square

goodness-of-fit statistic exceeds 10, the uncertainty of the auroral boundaries exceeds 1° for boundaries derived from WIC images, etc.

4.4 Auroral Electrojet Indices

The Auroral Electrojet (AE) indices were downloaded from the World Data Center (WDC) at Kyoto in Japan. WDC provides these indices by processing data from different observatories. The AE index is calculated from 10-13 selected observatories. These observatories are situated in the northern hemisphere along the auroral zone. They provide the geomagnetic variations in the horizontal component, which is used to calculate the AE indices. The AE index is calculated as the difference between the upper AU index and the lower AL index. The AU index is the largest normalized data value among the data from all the stations at each given time (UT). The AL index is the smallest normalized data value for the same. For every station, the data is normalized by subtracting the average of all the data from the station on the five international quietest days for each month, from each value of 1-minute data obtained at the station during that month. The AO index is the mean value of the AL and the AU indices, and all these four indices (AL, AU, AE, and AO) are collectively known as the AE indices. The AU and AL indices are an expression of the strongest current intensity of the eastward and westward auroral electrojets, respectively. The AE index describes the overall activity of the electrojets, and the AO index provides a measure of the equivalent zonal current.

The AE indices require data from observatories in the X, Y, Z coordinate system for digital stations. To trust these observations, many observatories are required. The X and Y components are converted to the H component by $H = \sqrt{X^2 + Y^2}$ to make the data compatible with other stations. Fort Churchill, Poste-de-la-Baleine, and Yellowknife are the three digital stations that provide data in the X, Y, Z coordinate sytem. There are twelve observatories used here, out of which seven of them are digital stations. The original digital H component data are used for the Barrow, College, Narsarsuaq, and Lerivogur digital stations. The Abisko data digitized from analog records at the station is used. For Dixon, Cape Chelyuskin, Tixie, and Cape Wellen, the H values were digitized at the Arctic

and Antarctic Research Institute in Petersburg.

We use the AL index to drive our model to obtain the auroral oval boundaries by comparing it to the British Antarctic Survey data. The AE indices are available for every 1-minute interval at the WDC. Daily plots and hourly values of the AE indices and contributing station plots give additional information on the indices.

4.5 Proposed Model

We calculate the corrected geomagnetic coordinates of the co-latitudes of the equatorward and the poleward auroral oval boundaries in units of degrees of latitude using the simple Fourier expansion series in cosine.

$$\theta = A_0 + A_1 cos[15(t + A_2)] + A_3 cos[15(2t + A_4)] + A_5 cos[15(3t + A_6)] \tag{4.4}$$

A_0, A_1, A_3, and A_5 are amplitudes expressed in degrees of latitude. t is the local time in hours. A_2, A_4, and A_6 are the phases also given in hours. Starkov discusses the calculation of the sizes of the ovals in detail [19]. The variations of the size of the polar cap and the auroral oval are calculated by obtaining a formula for the surface area confined by the curve of equation (4.4) in the polar coordinates. This surface area is equal to one half of the integral from $r^2 d\varphi$. After integration, the surface area is obtained as

$$S = \pi \sum_{i=0,1,3,5} A_i^2. \tag{4.5}$$

The amplitude A_0 is at least 5-20 times greater than that of A_1, and A_1 is greater than the subsequent harmonics. Therefore, A_0 primarily defines the surface area. A comparison of the equation (4.5) with the AL index shows confirms that the luminosity inside the oval gets closer to its poleward boundary at the beginning of the auroral substorm development, and also the size of the polar cap equals the area of the high-latitude region inside the oval during strong magnetic activity. The latitude is given by $(90 - \theta)$. The coefficients A_i for i ϵ [0, ... , 6], are derived by the formula

$$A_i = b_0 + \frac{b_1}{2}\left[1 + tanh\left(\frac{AL - \frac{max(AL)}{2}}{b_2 \cdot mean(AL)}\right)\right]. \tag{4.6}$$

The b_0 controls the offset of the function. Increasing or decreasing b_0 moves the curve higher or lower, respectively. b_1 is an offset and amplitude control parameter, centering the function at $\frac{b_1}{2}$. The value of b_1 proportionately varies the peak-to-peak amplitude as well as the offset of the function. b_2 varies the slope of the function, making it steeper or smoother proportionately with respect to its value. This variation is translated to the peak-to-peak amplitude variation of the curve.

The hyperbolic tangent function is used to fit the data because this function gives a better estimation of the boundaries than a regression approach which uses approximation of polynomials of different degrees. The *tanh* function allows better control and flexibility in obtaining the boundaries, as the control parameters can be easily adjusted to make the function lie inside the limits. Also, this function does not allow the boundaries to go beyond the peak-to-peak amplitudes, thereby confining them within realistic values. The hyperbolic tangent function looks like Fig. 4.7. To demonstrate the elegance of the approach, the following considerations and steps were carried out.

1. The March 22nd, 2002 storm was chosen.
2. The chosen storm period was between March 22nd to March 29th, the period when storm activity was high.
3. The model was run through the DE algorithm for 1000 iterations.
4. The data was obtained from the British Antarctic Survey website: `http://www.antarctica.ac.uk/bas_research/our_research/az/magnetic_reconnection/wic_boundary_data.html`.
5. The AL indices were obtained from the Data Analysis Center for Geomagnetism and Space Magnetism of Kyoto University.
6. There are 4278 ovals in total during this storm.

7. In this case, the equation (4.4) has been approximated to just the first term A_0, as this gives a rough estimate of the radius of the oval and the other terms are higher harmonics of the expansion, which could be ignored for this run.

8. In every iteration, the boundaries of all the 4278 ovals are calculated individually using the model, for both the pole-ward and equator-ward edges. Then, the errors between the data and the model output are calculated. The sum square is calculated for every oval, and the maximum value is minimized in every loop.

9. The results were compared against the Starkov model output, at the March 22nd, 03:08:20 oval. The Kp index is 3 at this time.

10. The reason why this particular time was chosen is because this was one of the instances where there was substantial data available on the 24-hour MLT.

4.6 Results

Carrying off from the previous setup, the experiment can be visualized as fitting a suitable circle to the auroral oval data available on March 22nd, at 03:08:20. The proposed model does remarkably well in fitting the circle to the available data, when the DE algorithm is trained on them for 1000 generations for the whole storm. Figure 4.8 and fig. 4.9 show the auroral oval boundaries at this particular instance, at the equator-ward and pole-ward edges

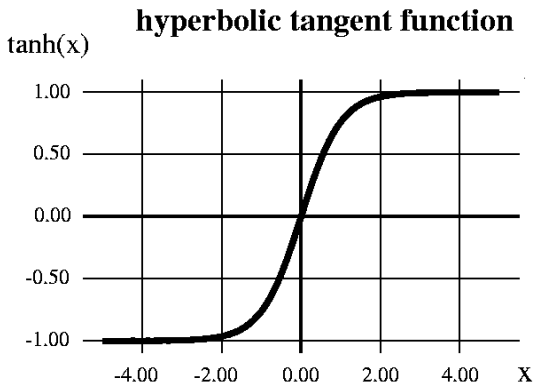

Fig. 4.7: Hyperbolic tangent function.

respectively. Figure 4.10 and fig. 4.11 show the convergence plots for these two cases. In both instances, there were steep convergences that occurred, within about 25 generations. The fitness values attained in both cases were about 1500.

Next, the proposed model was run through the DE algorithm, by considering all the coefficients of equation (4.4) in play. The expansion coefficients for this model are documented in Table 4.3. Figure 4.12 and fig. 4.13 show the auroral oval boundaries, once again, for March 22nd, at 03:08:20, at the equator-ward and pole-ward edges, respectively. Figure 4.14 and fig. 4.15 show the convergence plots for these two cases. In both instances, there were steep convergences that occurred, within about 100 generations. The fitness values attained in both cases were about 1300.

The proposed model was built on higher resolution parameters compared to the model built by Feldstein-Starkov, and caters to more dynamic responses, especially during active

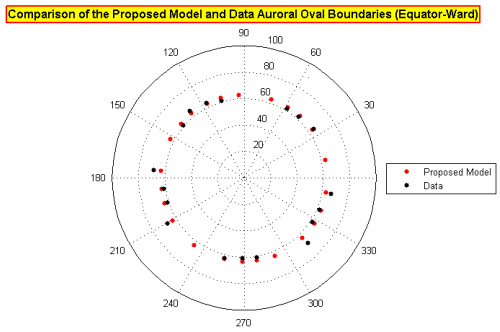

Fig. 4.8: A_0 coefficient fit to the equator-ward boundary.

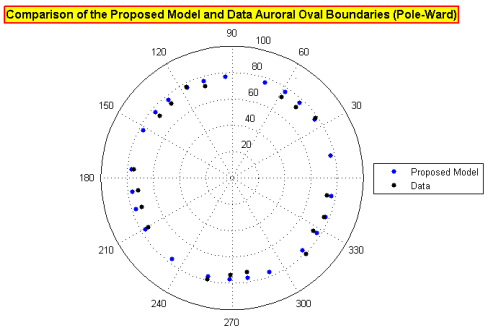

Fig. 4.9: A_0 coefficient fit to the pole-ward boundary.

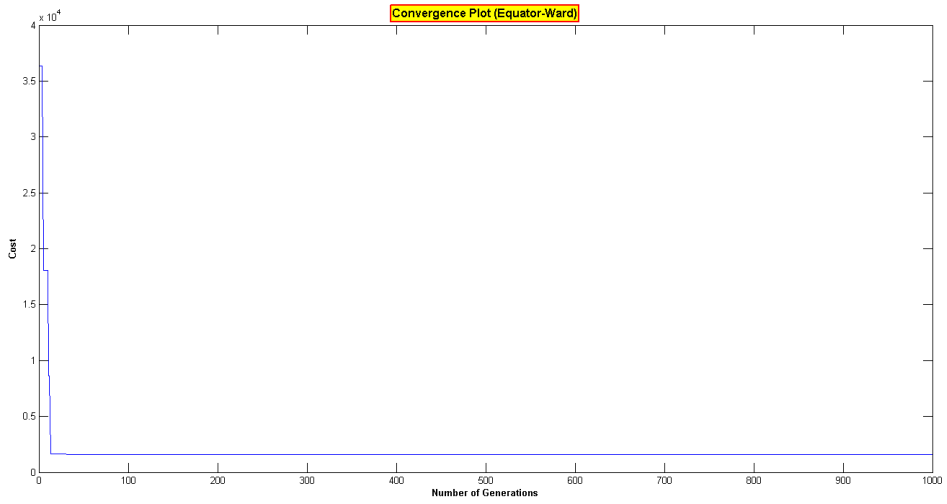

Fig. 4.10: Convergence plot for the A_0 coefficient fit to the equator-ward boundary.

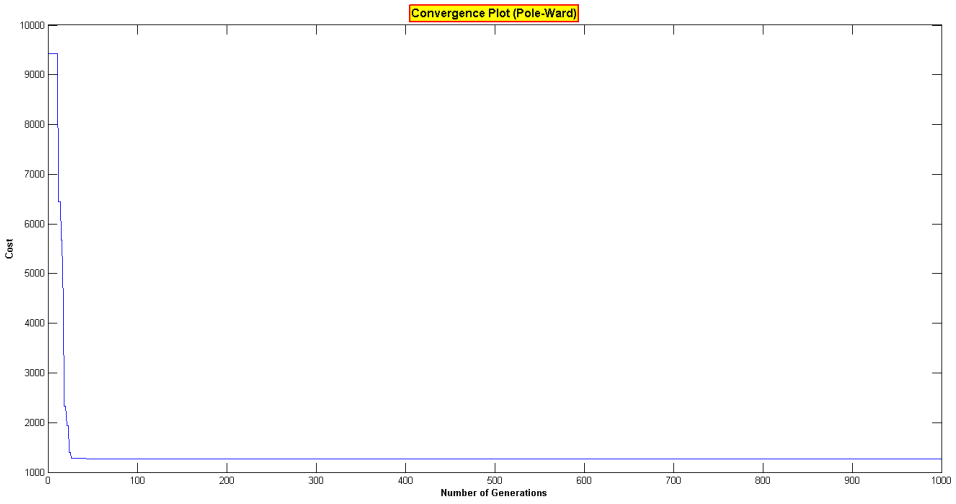

Fig. 4.11: Convergence plot for the A_0 coefficient fit to the pole-ward boundary.

geomagnetic storms. The data used for training the model against is also newer than the data used for the Feldstein-Starkov model. This is a strong model which can be used to model and study the morphology of the auroral oval boundaries quite efficiently. There is a clear thickening of the auroral oval bulge as the storm reaches its maximum strength.

Table 4.3: Proposed model expansion coefficients for the auroral boundaries.

(m = 0) Pole-ward boundary of the auroral oval							
	A_{00}	A_{10}	A_{20}	A_{30}	A_{40}	A_{50}	A_{60}
b_{00}	13.3033	1.0022	86.4289	1.1344	90.4790	2.4300	28.3775
b_{10}	7.0451	1.3190	22.4314	3.5589	69.6531	2.9365	46.2638
b_{20}	5.5638e-10	1.3464e-10	7.6775e-10	2.9761e-10	8.8439e-10	1.5903e-10	2.4561e-10
(m = 1) Equator-ward boundary of the auroral oval							
	A_{00}	A_{10}	A_{20}	A_{30}	A_{40}	A_{50}	A_{60}
b_{00}	26.0624	2.4707	52.2986	1.1738	58.8699	1.0561	49.2121
b_{10}	4.3866	6.3771	19.4890	4.3161	22.2436	6.5135	39.6764
b_{20}	4.1993e-10	8.2794e-10	1.4267e-10	8.6282e-12	3.8577e-10	5.7791e-10	2.2697e-10

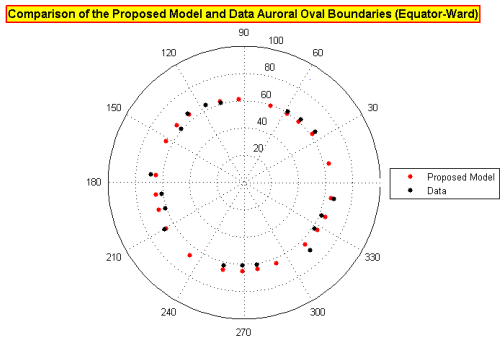

Fig. 4.12: Model fit to the equator-ward boundary.

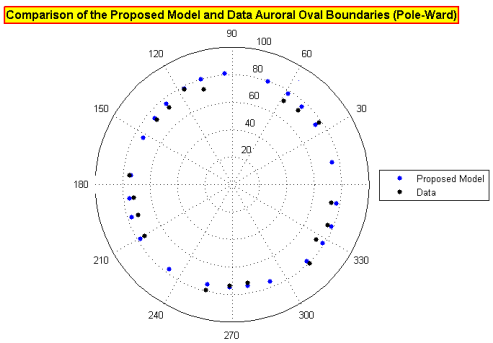

Fig. 4.13: Model fit to the pole-ward boundary.

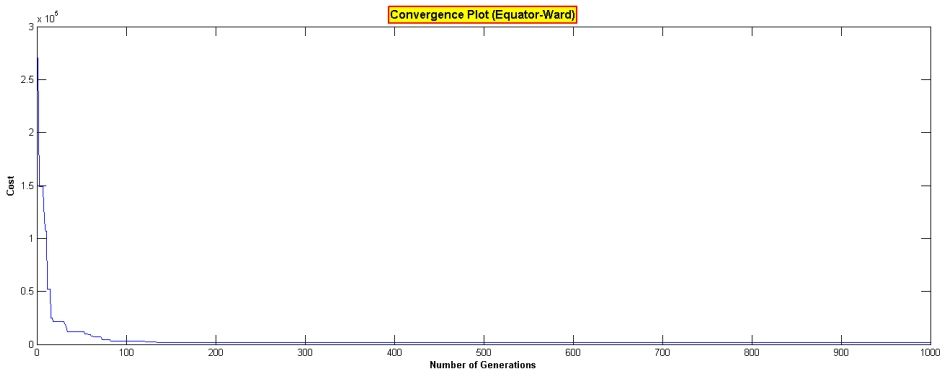

Fig. 4.14: Convergence plot for the model fit to the equator-ward boundary.

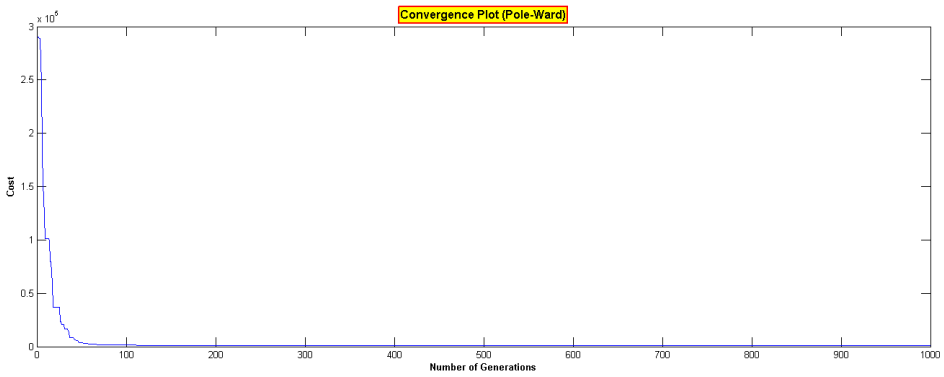

Fig. 4.15: Convergence plot for the model fit to the pole-ward boundary.

Chapter 5

Conclusions and Future Work

This thesis has presented the fact that the DE algorithm is a vibrant and active field of multi-disciplinary research in the optimization world. DE emerged as a simple and efficient scheme for global optimization more than a decade ago. Its simple and straightforward strategy, along with its ability to optimize a wide variety of multi-dimensional, multi-objective and multi-modal optimization problems, has made DE one of the most popular optimization algorithms as of today.

A comparative study was carried out on the RGA, PSO, and DE algorithms, to highlight the pros and cons of each of them. This benchmarking was done against six standard test functions, viz., Tripod function, Alpine function, Parabola function, Griewank function, Rosenbrock function, and Ackley function, in multiple dimensions. In all instances, the DE outperformed its counterparts in terms of convergence to the global minimum. The attained fitness value, too, was lower than the others. The simplicity and robustness of the DE algorithm was, however, offset by the run time of the algorithm. The PSO and RGA are faster algorithms.

The DE algorithm was used to build an auroral oval boundary model, which can be used to forecast auroral displays. This model was trained on data obtained from the British Antarctic Survey group, and the AL indices were obtained from the Data Analysis Center for Geomagnetism and Space Magnetism of Kyoto University. This proposed model was compared against the Feldstein-Starkov model. It was observed that this model showed very good adherence to the data, which is of 2-minute resolution approximately.

Once a forecast model was established, as an extended study, the efficiency and diversity of the DE was further put to test, by applying it to the phase-locked loop circuits, which is discussed in the appendix section. The LPLL circuit was chosen, and the DE algorithm was

used to train the coefficients of the first-order lowpass filter and the second-order loop filter, against the best VCO input signal response curve. This signal was generated synthetically to simulate the most ideal response when there is a step in frequency. The DE algorithm was also applied on the standard IC 74HC/HCT297 ADPLL circuit, to obtain the characteristic parameters of the circuit, viz., K, M, N, and the center frequency f_0, when there was a step jump applied at the input.

The DE algorithm's method of self-organization is remarkable. The vector generation scheme in the algorithm leads to a fast increase of population vector distances if the objective function surface is flat. This property prevents DE from advancing too slowly in shallow regions of the objective function surface and allows for quick progress after the population has traveled through a narrow valley. If the vector population approaches the final minimum, the vector distances decrease due to selection, allowing the population to converge reasonably fast. But although there have been numerous variations implemented on the classic DE algorithm, it largely remains problem-specific. In other words, certain DE variants are more suited for a specific group of applications, and do not perform as good on others. Much scope remains, on understanding how to choose the control variables for different problems, if the combination or hybridization of DE with other algorithms would serve a greater purpose or not, etc.

The proposed auroral oval model in this thesis could be used to predict variations in auroral oval boundaries during storm periods with good accuracy. There are lots of possibilities for further improving the model. It was discovered that there is a huge chunk of missing data points in the historical database of the British Antarctic Survey group. The reasons could be because of the loss of data due to the nature of the satellite orbits, or other disturbances in space. The model could be vastly improved if more data is available, and if this data is of higher resolution.

References

[1] M. Clerc, *Particle Swarm Optimization.* London, Newport Beach: International Society for Technology in Education, 2006.

[2] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, MA: Addison-Wesley Publishing Company, 1989.

[3] K. V. Price, R. M. Storn, and J. A. Lampinen, *Differential Evolution.* Berlin, Heidelberg, New York: Springer, 2005.

[4] K. Sastry, D. Goldberg, and G. Kendall, "Genetic algorithms," *E. K. Burke and G. Kendall (Eds.), Search methodologies: introductory tutorials in optimisation and decision support techniques, Berlin: Springer*, pp. 97–125, 2005.

[5] H. qi Li and L. Li, "A novel hybrid real-valued genetic algorithm for optimization problems," *International Conference on Computational Intelligence and Security*, 2007.

[6] J. Kennedy and R. Eberhart, "Particle swarm optimization," *Proceedings of IEEE International Conference on Neural Networks (ICNN95)*, pp. 1942–1948, 1995.

[7] Y. Shi and R. C. Eberhart, "A modified particle swarm optimizer," *Proceedings of IEEE International Conference on Evolutionary Computation, Anchorage, AK*, 1998.

[8] R. C. Eberhart and Y. Shi, "Comparison between genetic algorithms and particle swarm optimization," *Annual Conference on Evolutionary Programming, San Diego*, 1998.

[9] K. V. Price, "Differential evolution vs. the functions of the 2nd iceo," *Proceedings of IEEE International Conference on Evolutionary Computation*, pp. 153–157, 1997.

[10] K. V. Price and R. Storn, "Differential evolution: A simple evolution strategy for fast optimization," *Dr. Dobbs Journal*, vol. 22, pp. 18–24, 1997.

[11] R. Storn and K. V. Price, "Differential evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces," *International Conference on Swarm Intelligence, USA, Technical Report. TR-95-012*, 1995.

[12] K. V. Price and R. Storn, "Minimizing the real functions of the icec 1996 contest by differential evolution," *Proceedings of IEEE International Conference on Evolutionary Computation*, pp. 842–844, 1996.

[13] R. Storn, "On the usage of differential evolution for function optimization," *Proceedings of North American Fuzzy Information Processing Society*, pp. 519–523, 1996.

[14] S.-I. Akasofu, "The latitudinal shift of the auroral belt," *Journal of Atmospheric and Terrestrial Physics*, vol. 26, pp. 1167–1174, 1964.

[15] Y. Feldshteyn, "Some problems concerning the morphology of auroras and magnetic disturbances at high latitudes," *Geomagnetism Aeronomy*, vol. 3, pp. 183–92, 1963.

[16] Y. Feldstein, "Auroral oval," *Journal of Geophysical Research*, vol. 78, pp. 1210–1213, 1973.

[17] Y. Feldstein and G. Starkov, "The auroral oval and the boundary of closed field lines of geomagnetic field," *Planetary and Space Science*, vol. 18, pp. 501–508, 1970.

[18] R. Holzworth and C.-I. Meng, "Mathematical representation of the auroral oval," *Geophysical Reseach Letters*, vol. 2, pp. 377–380, 1975.

[19] G. Starkov, "Mathematical model of the auroral boundaries," *Geomagnetism and Aeronomy*, vol. 34, pp. 331–336, 1994.

[20] M. Gussenhoven, D. Hardy, and N. Heinemann, "Systematics of the equatorward diffuse auroral boundary," *Journal of Geophysical Research*, vol. 88, pp. 5692–5708, 1983.

[21] D. Hardy and M. Gussenhoven, "A statistical model of auroral electron precipitation," *Journal of Geophysical Research*, vol. 90, pp. 4229–4248, 1985.

[22] D. Hardy, M. Gussenhoven, R. Raistrick, and W. McNeil, "Statistical and functional representations of the pattern of auroral energy flux, number flux, and conductivity," *Journal of Geophysical Research*, vol. 92, pp. 12 275–12 294, 1987.

[23] D. Hardy, M. Gussenhoven, and D. Brautigam, "A statistical model of auroral ion precipitation," *Journal of Geophysical Research*, vol. 94, pp. 370–392, 1989.

[24] Y. Zhang and L. Paxton, "An empirical kp-dependent global auroral model based on timed/guvi fuv data," *Journal of Atmospheric and Solar-Terrestrial Physics*, vol. 70, pp. 1231–1242, 2008.

[25] J. Bartels, N. Heck, and H. Johnston, "The three-hour-range index measuring geomagnetic activity," *Terrestrial Magnetism and Atmospheric Electricity*, vol. 44, pp. 411–454, 1939.

[26] K. A. Costello, "Moving the rice msfm into a real-time forecast mode using solar wind driven forecast models," Ph.D. dissertation, Rice University, Houston, Texas, 1997.

[27] S. Wing, J. Johnson, J. Jen, C.-I. Meng, D. Sibeck, K. Bechtold, J. Freeman, K. Costello, M. Balikhin, and K. Takahashi, "Kp forecast models," *Journal of Geophysical Research*, vol. 110, pp. 411–454, 2005.

[28] G. Starkov, "Statistical dependences between the magnetic activity indices," *Geomagnetism and Aeronomy*, vol. 34, pp. 101–103, 1994.

[29] E. V. Appleton, "Automatic synchronization of triode oscillators," *Proceedings of Cambridge Philosophical Society*, vol. 21, p. 231, 1922.

[30] A. B. Grebene and H. R. Camenzind, "Phase locking as a new approach for tuned integrated circuits," *International Solid-State Circuits Conference Digest of Technical Papers*, pp. 100–101, 1969.

[31] R. E. Best, *Phase-Locked Loops: Design, Simulation and Applications.* Oberwil, Switzerland: McGraw Hill, 2007.

Appendix

Phase-Locked Loop Optimization

A.1 Introduction

The history of the phase-locked loop (PLL) dates back to as early as 1932. The PLL helps certain operations to function in an orderly manner. In the television, the PLL makes sure that the object is oriented right, i.e., the heads are at the top of the screen and the feet are at the bottom. A PLL is a circuit synchronizing an output signal (generated by an oscillator) with a reference or input signal in frequency as well as in phase. In the locked or synchronized state, the phase error between the oscillator's output signal and the reference input signal is zero, or remains a constant [29].

PLLs are widely employed in radio, telecommunications, computers, and other electronic applications. They can be used to recover a signal from a noisy communication channel, generate stable frequencies at a multiple of an input frequency (frequency synthesis), or distribute clock timing pulses in digital logic designs such as microprocessors. The four main types of PLLs are Linear PLL (LPLL), Digital PLL (DPLL), All-Digital PLL (ADPLL), and Software PLL (SPLL). The PLL consists of the following three basic functional blocks [30]:

1. A voltage-controlled oscillator (VCO),
2. A phase detector (PD),
3. A loop filter (LF).

In the following sections, we shall mainly focus on using the DE algorithm on the LPLL and the ADPLL. These two cases were chosen because they serve as good representatives for nonlinear problems in the analog and digital domains, respectively. In the case of LPLL, the DE algorithm was used to train the loop filter coefficients to get the optimum phase-locked loop response of a bandpass analog PLL with 5 kHz center frequency and 100 Hz

loop bandwidth with a 100 Hz frequency step applied at a certain time instance. In the case of ADPLL, the IC 74HC/HCT297 circuit was first built in Simulink and the output response was studied. The whole circuit was then modelled behaviorally, by making every block in the circuit a behavior model, and the DE algorithm was run on it to train the parameters that describe the circuit when a frequency step is applied at a particular time instance.

A.2 Linear Phase-Locked Loop

A.2.1 Background

The LPLL or analog PLL is the classical form of PLL. All components in the LPLL operate in the continuous-time domain. An LPLL block diagram is shown in Fig. A.1.

The input signal $u_1(t)$ is usually a sine wave with angular frequency ω_1 [31]. The output signal $u_2(t)$ is a symmetrical square wave of angular frequency ω_2. In the locked state, the two frequencies are equal. The output signal $u_d(t)$ of the phase detector consists of a dc component and is roughly proportional to the phase error θ_e. The remaining terms are ac components having frequencies of $2\omega_1$, $4\omega_1$, and so on. The loop filter, which is a low-pass filter, removes these unwanted higher frequencies. The design of an LPLL follows the following sequence of steps.

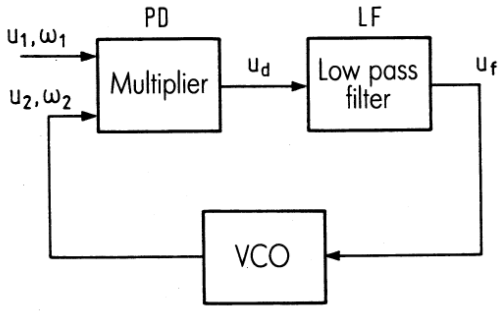

Fig. A.1: Block diagram of the LPLL.

1. Specify the center (angular) frequency ω_0. It could be a constant, or a range to lock onto different channels.

2. Specify the damping factor ζ. It should be chosen to obtain fast response of the LPLL onto phase or frequency steps applied to its reference input.

3. Specify the lock range $\Delta\omega_L$ or the noise bandwidth B_L. The natural frequency ω_n depends on these two factors.

4. Specify the frequency range of the LPLL. If the VCO output signal has a minimum frequency of ω_{2min} and a maximum frequency of ω_{2max}, it must be guaranteed that $\omega_{2min} < \omega_{0min} - \Delta\omega_L$ and $\omega_{2max} > \omega_{0max} + \Delta\omega_L$.

5. The VCO gain K_0 is calculated as $K_0 = \frac{\omega_{2max} - \omega_{2min}}{u_{fmax} - u_{fmin}}$, where u_{fmin} and u_{fmax} are the minimum and maximum values allowed for u_f, respectively, which is the control signal.

6. Determine the phase detector gain K_d.

7. Determine the natural frequency ω_n, which can be calculated either as $\omega_n = \frac{\Delta\omega_L}{2\zeta}$, or as $\omega_n = \frac{2B_L}{\zeta + 1/4\zeta}$.

8. Select the type of loop filter and calculate the time constants τ_1 and τ_2.

9. Determine the values of R_1, R_2, and C of the loop filter using the equations $\tau_1 = R_1 C$ and $\tau_2 = R_2 C$.

A.2.2 Simulation

A bandpass LPLL with 5 kHz center frequency and 100 Hz loop bandwidth was modelled in Simulink. Figure A.2 shows this implementation. A 100 Hz frequency step is applied to the input at 40 ms. A synthetic signal was generated to represent an ideal VCO input in response to a 100 Hz frequency step at 40 ms. The first-order low-pass filter following the multiplier phase detector is used to remove any noise that could possibly occur on the

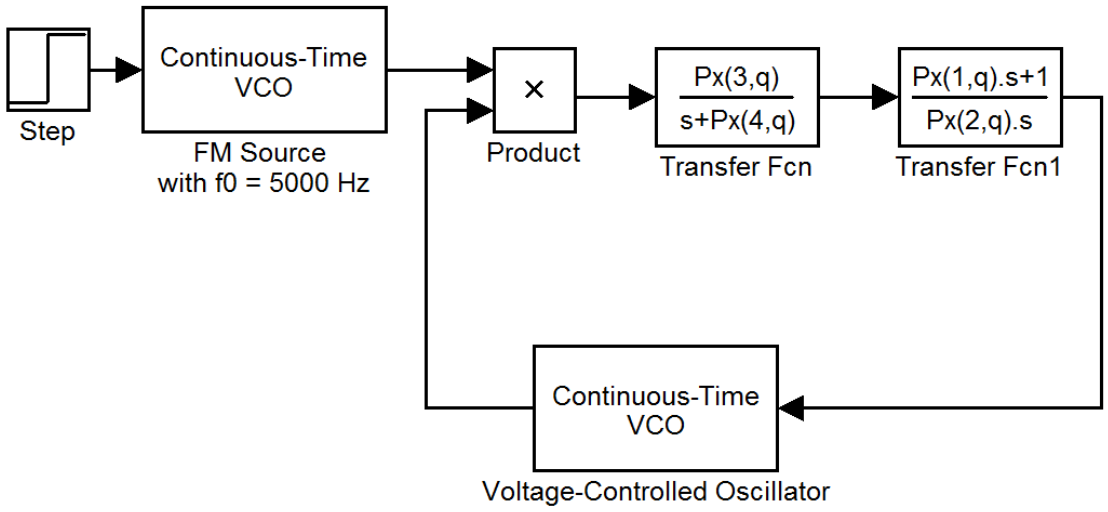

Fig. A.2: Simulink model of an LPLL at 5 kHz.

VCO control signal. The coefficients depicted in the transfer function blocks form the target vector population, and those are what are optimized in the DE algorithm. These are the coefficients of the first-order low-pass filter and the second-order loop filter.

A.2.3 Results

The cost function would be the input signal to the VCO, and it tries to match against the synthetic signal itself, to carry out the optimization of these coefficients. This synthetic signal is shown in Fig. A.3. As can be seen, there is a frequency step applied at 40 ms, and the loop settles down at about 100 ms again. The DE algorithm is initialized with a population size of 8, and is run for 1000 generations. The Simulink model is run in every generation, and the output of the loop filter, or the input signal to the VCO is the tracking signal. This tracking signal is exported into the Matlab environment, and interpolation is carried out onto the trial and the target vectors in every generation, to match the exact number of data points the synthetic signal carries.

The second-order loop filter transfer function is of the form $\left(\frac{a_1 \cdot s+1}{a_2 \cdot s}\right)$, and the first-order lowpass filter transfer function is of the form $\left(\frac{a_3}{s+a_4}\right)$. The obtained results are presented in Table A.1. These coefficients are compared to the values that Dr. Mark A. Wickert of

University of Colorado presented in his lectures. These coefficients are shown in Table A.2. As can be seen, these values are very close.

Figure A.4 shows the comparison between the synthetic signal and the model signal after 1000 generations. As can be seen, the fit obtained is pretty good, and one can be hardly differentiated from the other, inspite of the intermediate interpolation carried out on the signals. Figure A.5 shows the convergence plot. There is an aggressive convergence within about 20 generations itself, and then the algorithm slowly tries to converge towards a lower value. The lowest value obtained in 1000 generations was 0.2091.

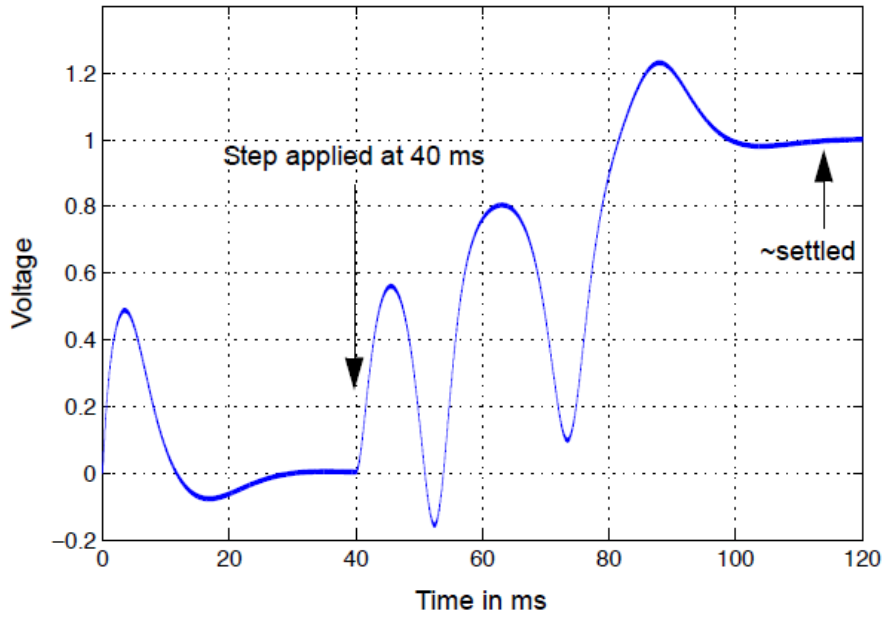

Fig. A.3: Synthetic VCO input signal.

Table A.1: Coefficients of the second-order loop filter and the first-order lowpass filter.

a_1	a_2	a_3	a_4
0.0070	0.0091	1129.3	1048.3

Table A.2: Coefficients presented by Dr. Mark A. Wickert.

a_1	a_2	a_3	a_4
0.0075	0.008836	1000	1000

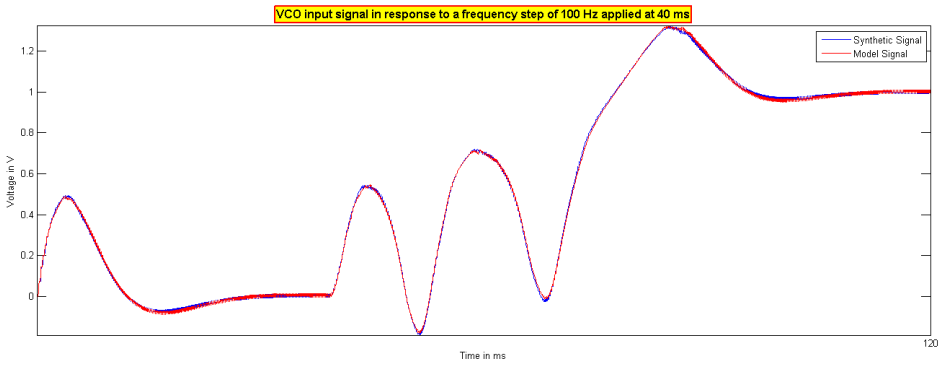

Fig. A.4: Comparison of the VCO input signals.

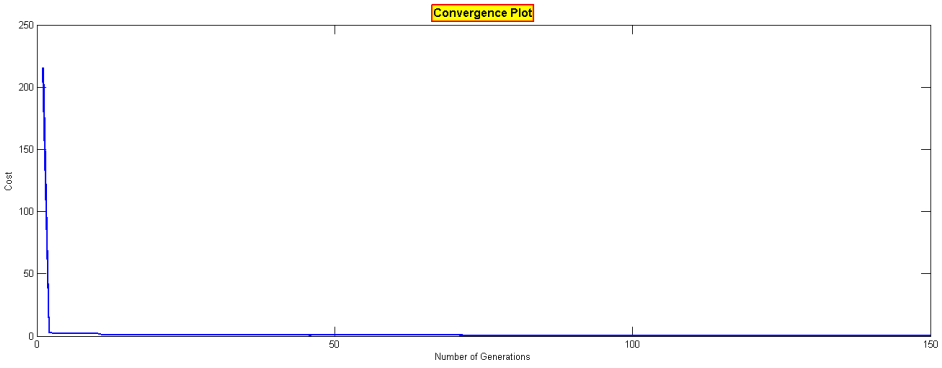

Fig. A.5: Synthetic VCO input signal.

A.3 All Digital Phase-Locked Loop

A.3.1 Background

The main functional block structure of the ADPLL circuit is pretty similar to the LPLL structure, but in this case, each of the functional blocks is entirely digital, or in other words, made up of only digital components. The ADPLL implemented in this section is the most often used configuration, and is based on the IC 74HC/HCT297, as shown in fig. A.6. The whole circuit was constructed in Simulink. The main blocks are discussed below.

All-digital phase detector - The phase detector used here is a simple EXOR gate. Figure A.7 depicts the waveforms for different phase errors θ_e. At zero phase error, the signals u_1 and u_2' are exactly out of phase by 90°, as shown in fig. A.7(a). The duty cycle is exactly 50% and the output signal u_d is a square wave whose frequency is twice the

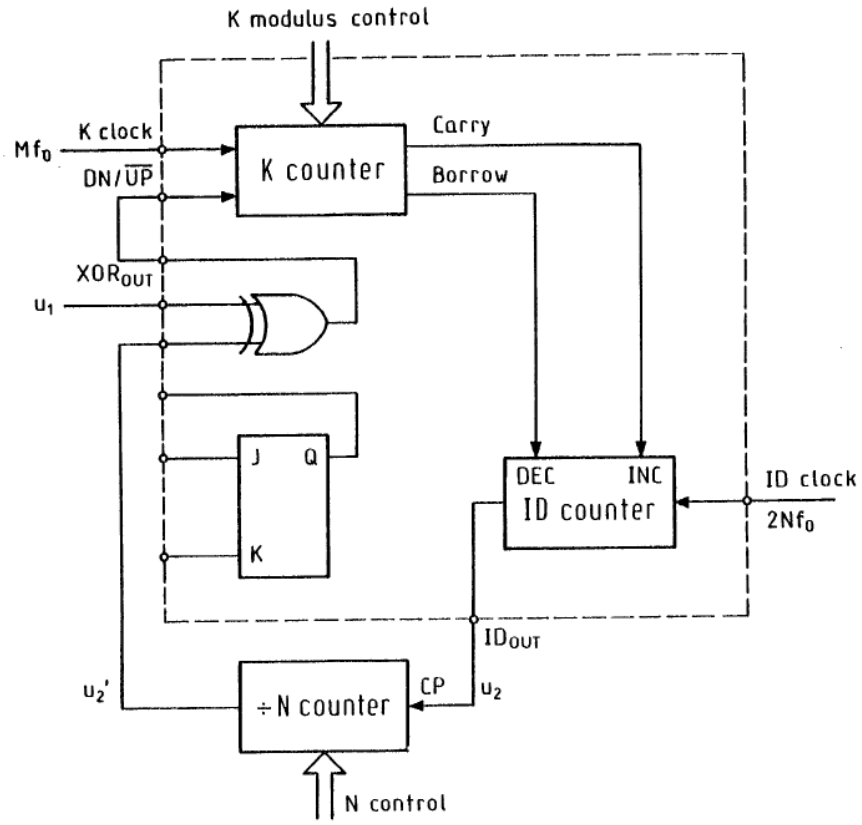

Fig. A.6: ADPLL configuration based on the IC 74HC/HCT297.

reference frequency. When there is a positive phase error, as shown in fig. A.7(b), then the duty cycle increases, and the reverse is true for the case where the phase error is negative. Therefore, the EXOR phase detector can maintain phase tracking when the phase error is confined to the range $-\frac{\pi}{2} < \theta_e < \frac{\pi}{2}$. Figure A.8 shows the Simulink model output of the phase detector.

All-digital loop filter - The loop filter used here is a K counter loop filter. It consists of two independent counters called UP counter and DN counter, as shown in fig. A.9, which are both counters that count upwards. K is the modulus of the both the counters, and is

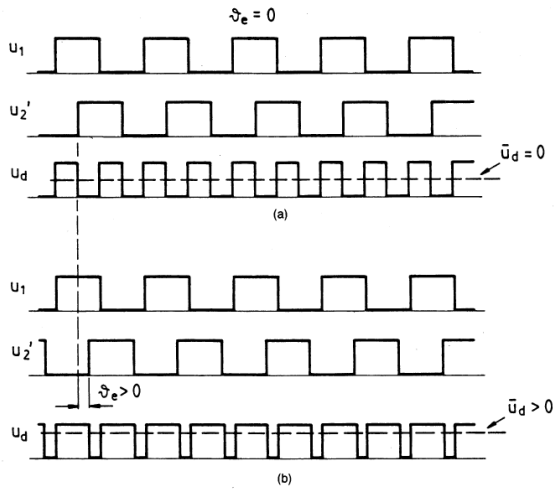

Fig. A.7: Waveforms for the EXOR phase detector. (a) Zero phase error, (b) Positive phase error.

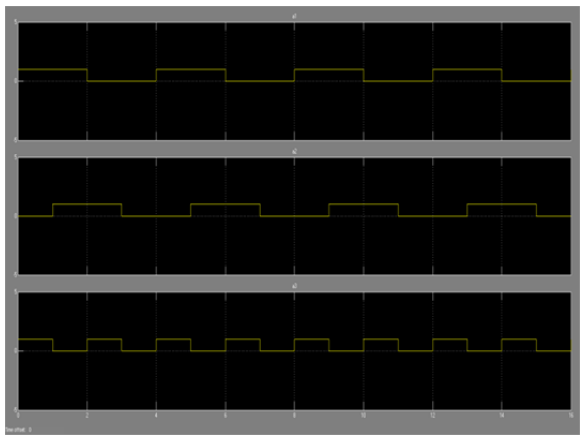

Fig. A.8: Simulink output of the EXOR phase detector.

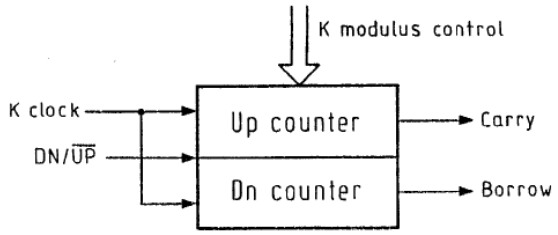

Fig. A.9: K counter loop filter.

always an integer of two. So both these counters count from 1 upto K-1. The frequency of the K clock is M times the center frequency f_0 of the ADPLL, where M is typically 8, 16, 32, etc. When the $DN/\bar{UP}$ signal is low, then the UP counter is active and the contents of the DN counter remains frozen. When it is high, then the DN counter is active and the contents of the UP counter remains frozen. The most significant bit of the UP counter is taken out as the carry output, and that of the DN counter is taken out as the borrow output. As a result, the carry signal goes high when the contents of the UP counter reaches or exceeds K/2, and the borrow signal does so when the contents of the DN counter reaches or exceeds K/2 as well. This is shown in fig. A.10. The digitally controlled oscillator is controlled by these carry and borrow signals. Figure A.11 is the circuit built in Simulink. Figure A.12 shows the Simulink model output of the flip flops of the UP counter. Figure A.13 shows the Simulink model output of the whole loop filter.

Digital-controlled oscillator (DCO) - The oscillator used here is the increment-decrement (ID) counter, as shown in fig. A.14. The operation of the ID counter can be understood from the waveforms shown in fig. A.15. The carry signal is fed into the INC input of the ID coutner, and the borrow signal to the DEC input. In the absence of these pulses, the ID counter divides the ID clock frequency by 2, as shown in fig. A.15(a). The circuit internally consists of a toggle flipflop. The signal at the inputs gets processed only when the toggle flipflop is set high. If the carry pulse at the INC input is "true" when the toggle flipflop is low, then it goes high onto the next positive edge of the ID clock. For the next two clock intervals thereafter, it stays low. This is shown in fig. A.15(b). If the carry pulse is "true" when the toggle flip flop is high, then it goes low during the next two

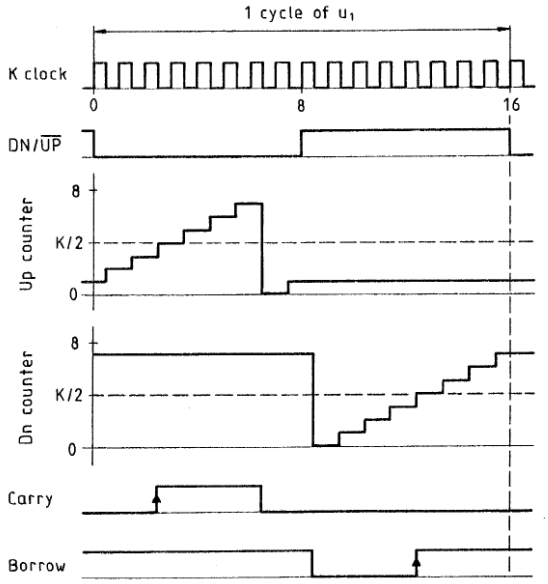

Fig. A.10: Waveforms for the K counter loop filter.

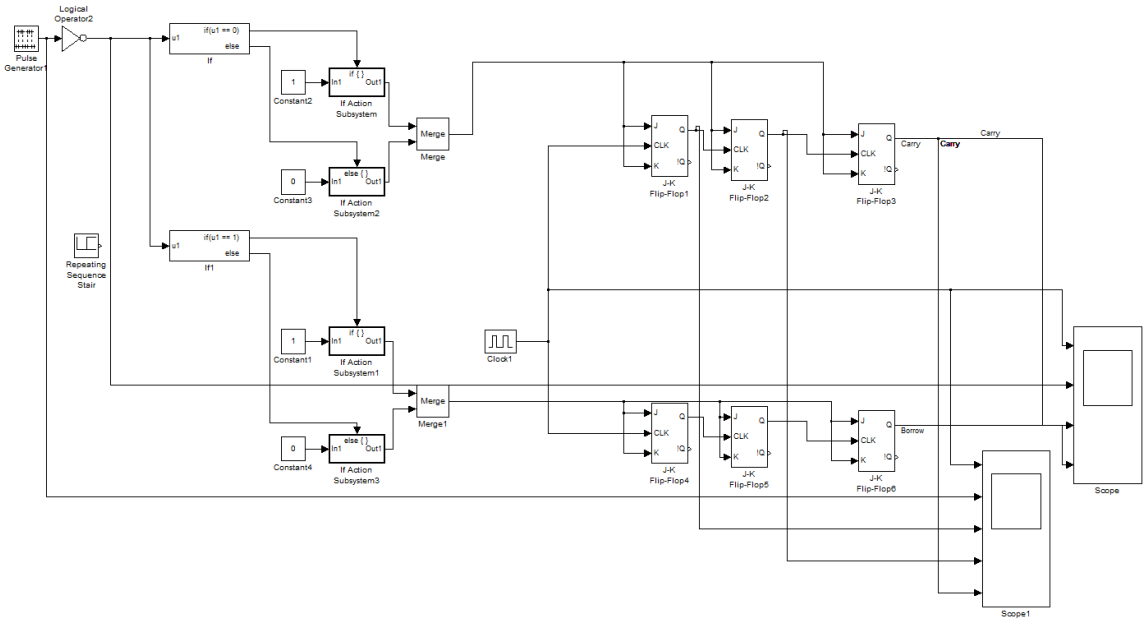

Fig. A.11: K counter loop filter circuit implementation in Simulink.

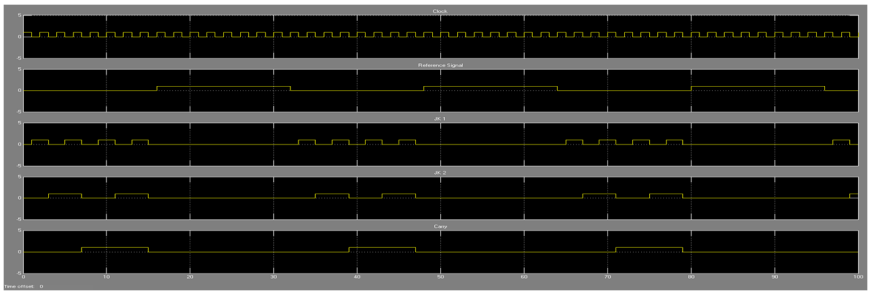

Fig. A.12: Simulink output of the flip flops of the UP counter.

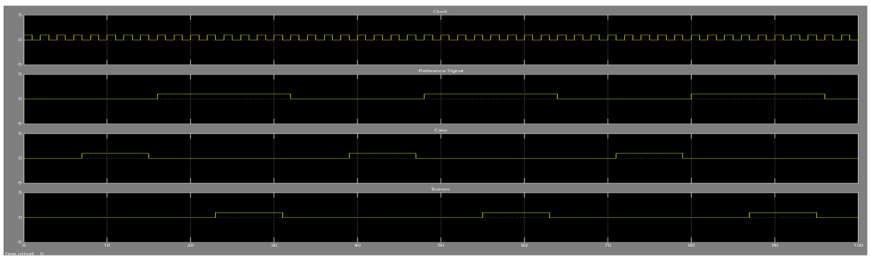

Fig. A.13: Simulink output of the K counter loop filter.

intervals of the ID clock. This can be seen in fig. A.15(c). As a result, the frequency at the output could increase upto two-thirds the value of the ID clock frequency. In analogy, when a borrow pulse arrives at the DEC input, it is processed only when the toggle flipflop is set low. This is shown in fig. A.15(d). Hence, there is a decrease in frequency at the output, which could go as low as one-thirds the value of the ID clock frequency. Figure A.16 is the circuit built in Simulink. Figure A.17 shows the Simulink model output of the DCO.

Divide-by-N counter - An external divide-by-N counter is used here, as shown by the circuit implementation in Simulink in fig. A.18. The ID counter clock frequency is 2N times that of the center frequency f_0 of the ADPLL. N is an integer power of 2 always, so the circuit divides the frequency of the input signal by that factor. Figure A.19 shows the

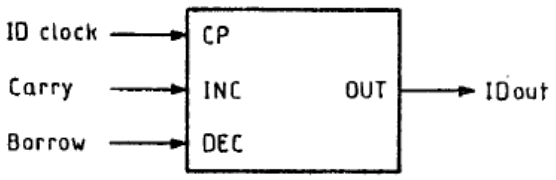

Fig. A.14: ID counter DCO.

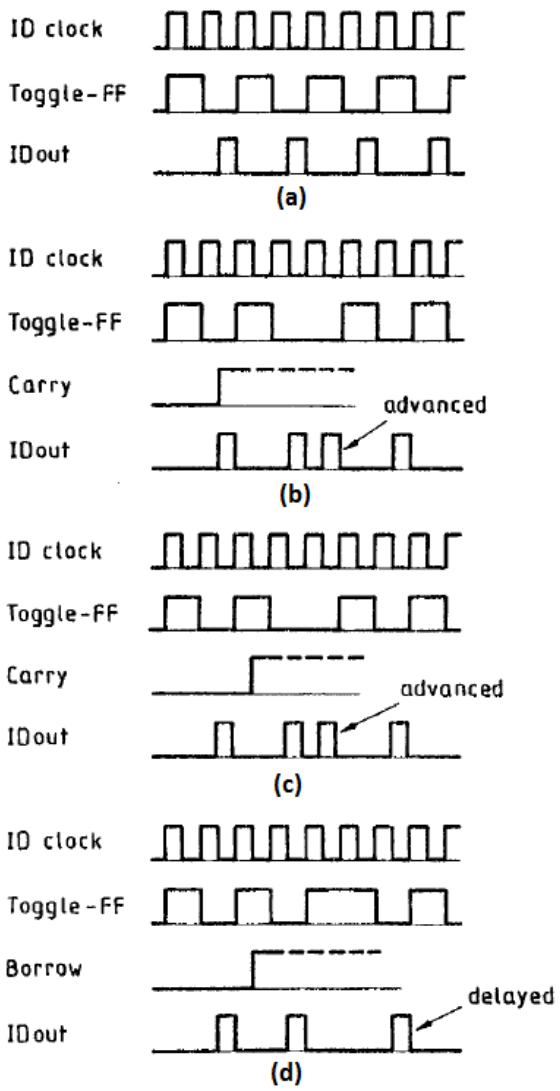

Fig. A.15: Waveforms for the ID counter DCO.

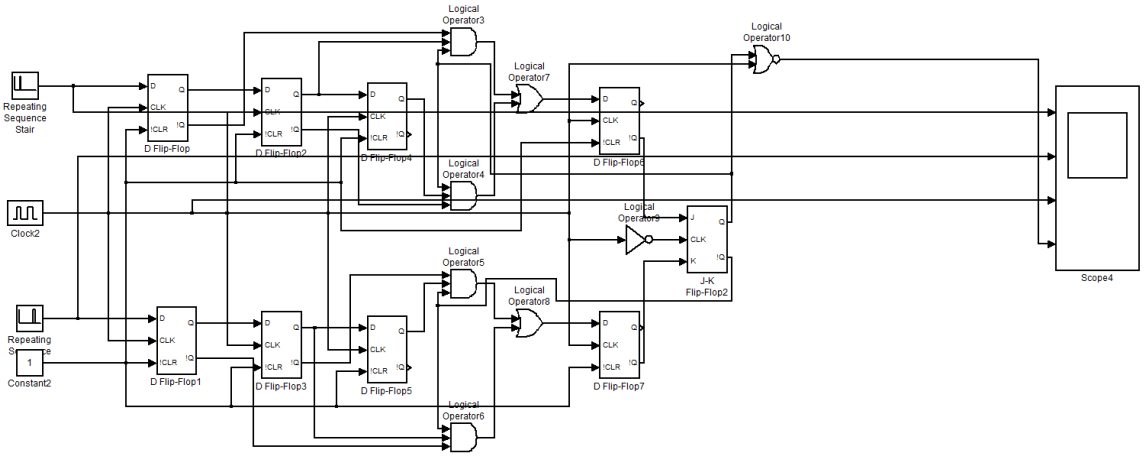

Fig. A.16: ID counter DCO circuit implementation in Simulink.

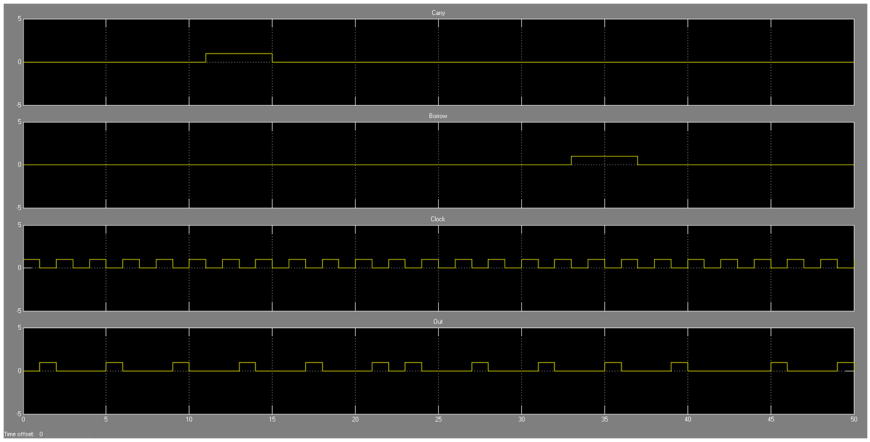

Fig. A.17: Simulink output of the digital-controlled oscillator.

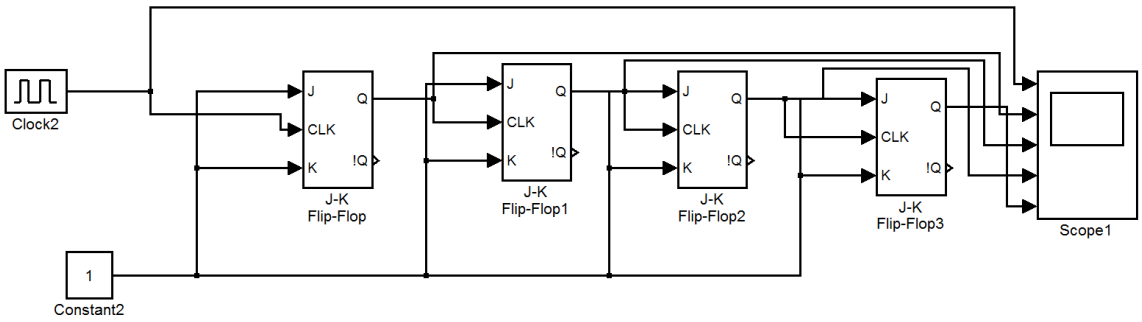

Fig. A.18: Divide-by-N circuit implementation in Simulink.

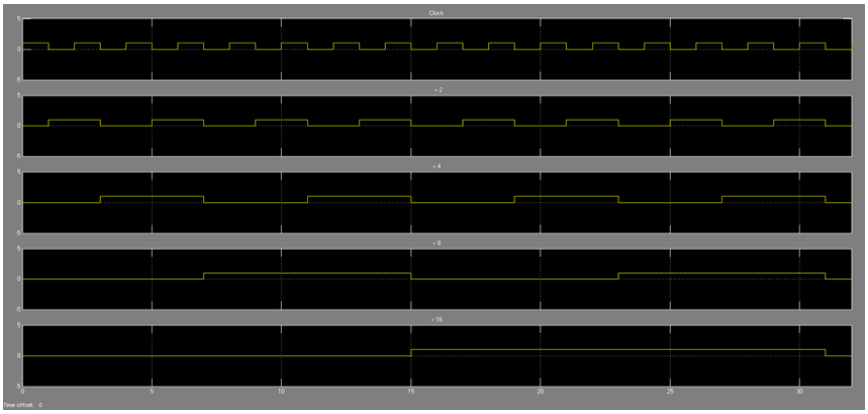

Fig. A.19: Simulink output of the divide-by-N counter.

Simulink model output of the DCO.

One of the major applications of this ADPLL configuration, is to build an ADPLL FSK decoder. Hence, to understand the design principle of the ADPLL, one can use the design methodology of an FSK decoder for convenience. One must bear in mind that this is a checklist, rather than a universal recipe. The steps are as follows.

1. Determine the center frequency f_0 of the ADPLL using $f_0 = \frac{f_1+f_2}{2}$ (this is an ideal approximation), and the hold range Δf_H of the ADPLL using $\Delta f_H = |f_1 - f_2|$.

2. If missing pulses exist in the reference signal, choose EXOR phase detector. Else, select JK-flipflop phase detector.

3. For minimum ripple, choose M = 4K for EXOR phase detector, M = 2K for JK-flipflop phase detector.

4. For simplicity, choose M = 2N.

5. Select K for the desired hold range using $\Delta f_H = f_0 \frac{M}{2KN}$. This gives $K = \frac{f_0}{\Delta f_H}$.

6. If $K < 8$, then M cannot be equal to 2N. So, choose the values for M back, given in step 3. Then, calculate the value of N given by $N = \frac{f_0 M}{\Delta f_H 2K}$, which gives $N = \frac{2f_0}{\Delta f_H}$ for EXOR phase detector, and $N = \frac{f_0}{\Delta f_H}$ for JK-flipflop phase detector. Then choose the minimum value of K, which is 8.

7. Calculate the remaining parameters.

8. Check if the hold range is within the allowed limits.

9. To avoid overslept carries and borrows, make sure that the condition $N > N_{min} = \frac{3M}{2K}$ is satisfied.

10. Check the settling time of the circuit, which is $\tau = \frac{KN}{2Mf_0}$ for EXOR phase detector, and $\tau = \frac{KN}{Mf_0}$ for JK-flipflop phase detector.

A.3.2 Simulation

We combine all the individual blocks discussed in the previous section together, as shown in fig. A.20, to form the ADPLL circuit of the IC 74HC/HCT297. Figure A.21 shows the Simulink model output of the ADPLL.

The simulink modules could not be used in the optimization scheme because the changing number of flip flops could not be accounted for. As a result, all the modules had to be modeled behaviorally. All signals need to have whole positive integers, so that the logic of the whole system could be carried out hierarchically. Behavioral modeling of the ADPLL was also performed for the purpose of comparison with the Simulink model. Every individual block was modelled behaviorally by building .m files for each. They were all connected to form the main ADPLL function by calling these sub-functions from the main file. The signals had to be modelled with a time stamp associated with every point. If there is a rise or a fall in amplitude of the signal, then it is considered to have zero delay as an ideal

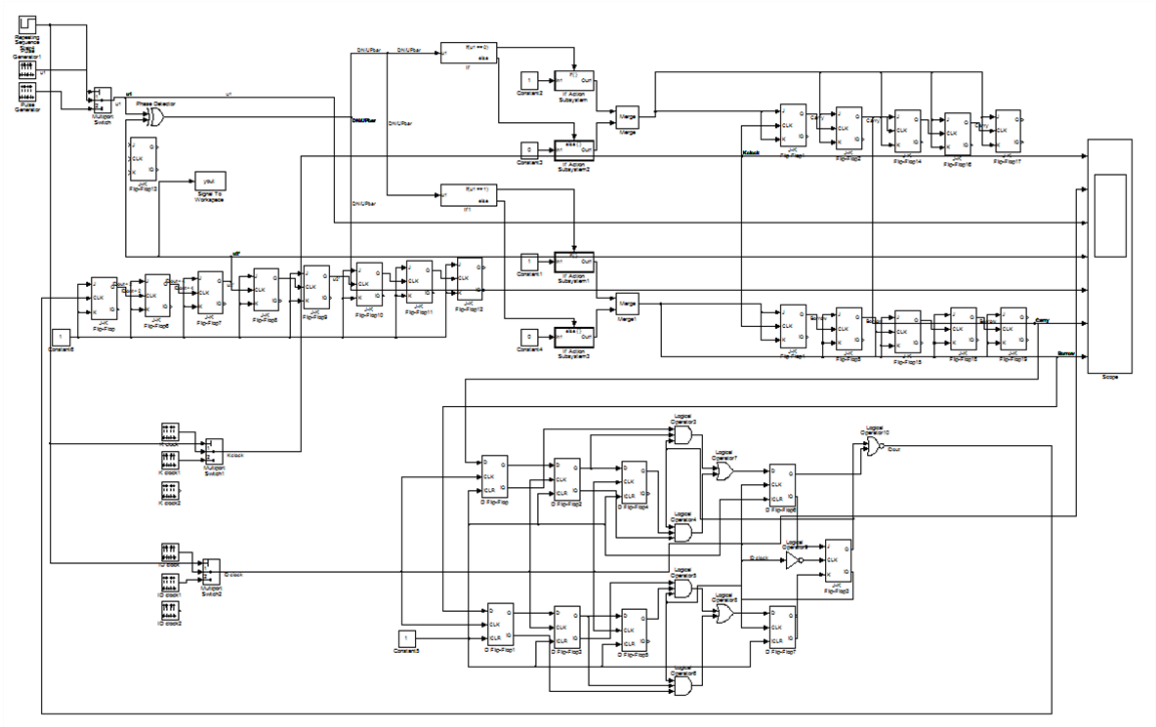

Fig. A.20: ADPLL circuit implementation in Simulink.

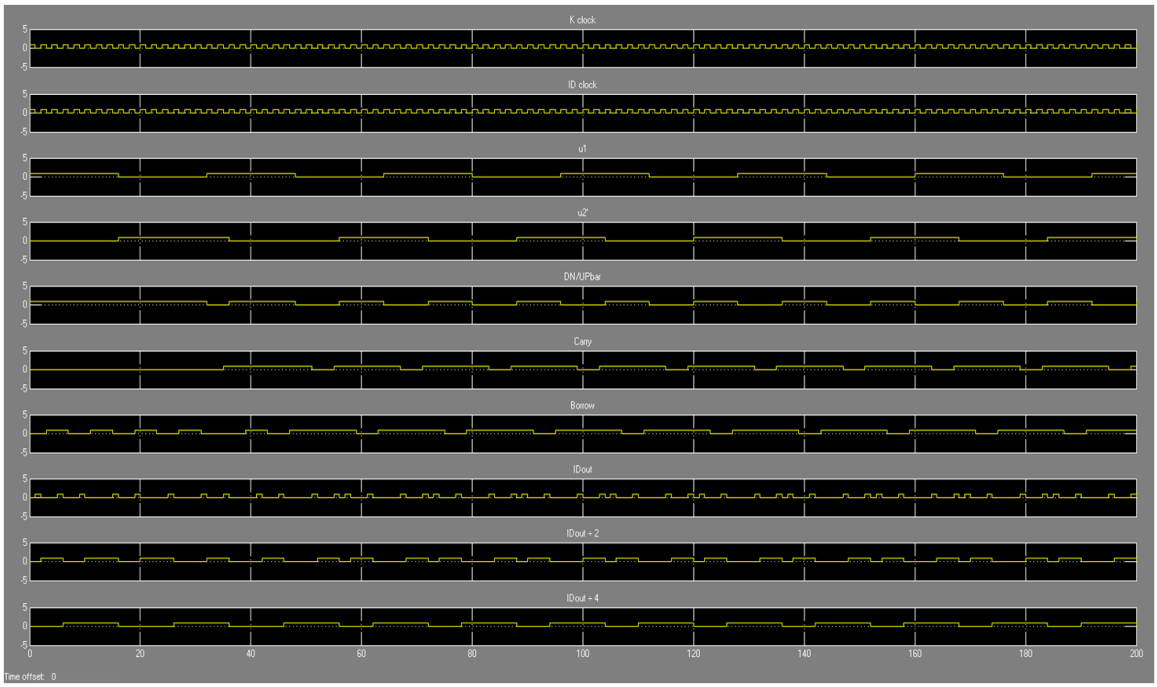

Fig. A.21: Simulink output of the ADPLL.

signal. At these locations, the time stamp is duplicated to represent both the amplitude values. Figure A.22 is a representation of such a signal. Table A.3 shows how the time stamping is done for the signal in fig. A.22.

The parameters to be optimized using the DE are f_0, K, M, and N. The generated u2' signal from every iteration was EXOR'ed with an ideal synthetic signal to give the error measure for the cost function.

A.3.3 Results

The DE was run for 500 generations with a population size fixed at 8. A step jump of 1 Hz to 5 Hz was applied at 600s. As can be seen in Fig. A.23, the algorithm fails to lock the ADPLL to two different frequencies. Rather, it produces a ripple effect at the output. Table A.4 shows the above mentioned parameters for this case presented by the author Roland E. Best in his book, and the parameters obtained by the algorithm after optimization. Due to the failure to lock to the different frequencies, the parameters are also way off as expected. But interestingly, the cost of the output produced by using the coefficients presented by Roland E. Best is 666, whereas that of the optimized case is 544, which is lesser. This suggests that, though the algorithm may fail to lock the circuit to different frequencies, it might as well produce in optimized parameters which could be used to build ADPLL's with lesser number of flip flops and reduce the overall cost of setup.

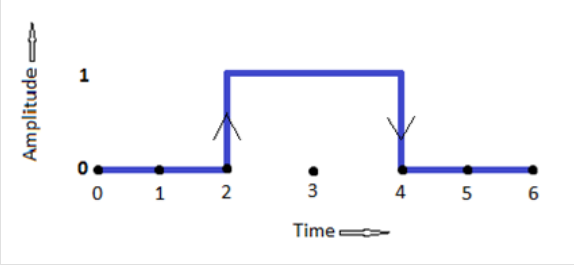

Fig. A.22: Signal representation for the behavior modelling.

Table A.3: Time and amplitude values of Fig. A.22.

Time	0	1	2	2	3	4	4	5	6
Amplitude	0	0	0	1	1	1	0	0	0

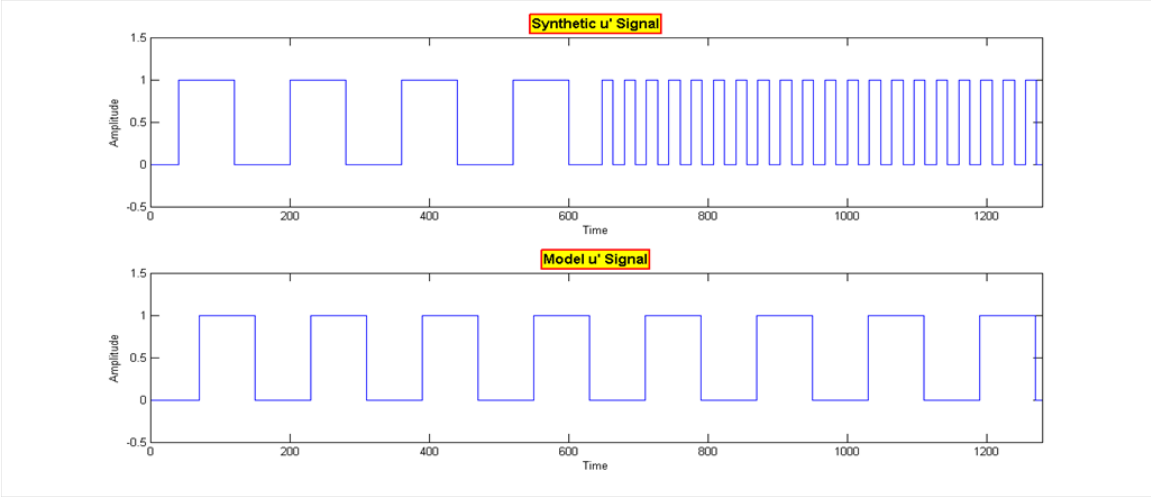

Fig. A.23: Diagram representing a failed attempt of an ADPLL locking scheme.

Table A.4: Optimization parameters.

	f_0	K	M	N
DE Algorithm	2	1024	3188	8
Roland E. Best	3	2	8	4